Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Informatyki

Projekt dyplomowy

X-ray Image Interpretation System

System wspomagający interpretację zdjęć rentgenowskich

Autorzy: Krzysztof Ćwiertnia, Kacper Kozik, Rafał Piwowar, Michał Sośnik
Kierunek studiów: Informatyka
Opiekun pracy: dr inż. Marcin Kuta

Kraków, 2025

Contents

Chapter 1

Project goals and vision

1.1. Problem definition

In today's world, the use of artificial intelligence in medicine is becoming increasingly popular. We are witnessing more cases where computers substitute humans in this field, aiding doctors in making diagnostic decisions. Numerous studies and projects are being developed to detect changes in the human body. This trend has motivated the writing of a thesis concerning the application of machine learning for interpreting X-ray images.

The objective of this thesis is to develop a system that assists doctors and users in detecting abnormalities in the human body based on X-ray images, such as fractures or cracks in selected parts of the skeletal system. The challenge involves creating deep learning models responsible for localizing defective body parts. The plan includes organizing datasets of X-ray images for selected types of bones and training a suitable deep learning model for each.

Such solutions often require substantial computational power and large datasets. Frequently, available datasets contain relatively few X-ray images suitable for training a robust machine learning model. In such cases, data augmentation is used to expand the dataset using existing images. The goal is to ensure the model's prediction quality and good generalization.

Another critical aspect of this project is maintaining the quality of the predictions, which is particularly vital in medical sciences. Consequently, appropriate metrics and techniques will be applied to enhance the accuracy of the developed system.

The final result of the project will be an application that enables users to upload X-ray images for interpretation to identify potential bone fractures. This application will be accessible to both medical professionals and general users, who can review analysis results and create accounts. General users will also have the ability to schedule consultations and communicate via video calls or chat with doctors through the platform. This creates another challenge, not only because the system and technologies need to be reliable, but also because it is important to ensure proper protection of personal data.

1.1.1. History of X-ray image

X-ray is a high-energy electromagnetic radiation. In many languages, it is referred to as Röntgen radiation, as it were discovered in 1895 by Wilhelm Conrad Röntgen (1845-1923) who was a Professor at Wuerzburg University in Germany [1].

Before X-rays were discovered in 1895, they were just a type of unidentified radiation emanating from experimental discharge tubes. The earliest experimenter thought to have (unknowingly) produced X-rays was William Morgan. In 1785, he presented a paper describing the effects of

passing electrical currents through a partially evacuated glass tube, producing a glow created by X-rays [2].

When Fernando Sanford created his "electric photography", he also unknowingly generated and detected X-rays. From 1886 to 1888, he became familiar with the cathode rays generated in vacuum tubes when a voltage was applied across separate electrodes [3].

Starting in 1888, Philipp Lenard conducted experiments to see whether cathode rays could pass out of the Crookes tube into the air. He found that something came through, that would expose photographic plates and cause fluorescence. He measured the penetrating power of these rays through various materials. It has been suggested that at least some of these rays were actually X-rays [4].

In 1889, Ivan Puluj published a paper on how sealed photographic plates became dark when exposed to the emanations from the tubes [5].

In 1894, Nikola Tesla noticed damaged film in his lab that seemed to be associated with Crookes tube experiments and began investigating this invisible, radiant energy. After Röntgen identified the X-ray, Tesla began making X-ray images of his own using high voltages and tubes of his own design, as well as Crookes tubes [6].

Working with a cathode-ray tube in his laboratory, Röntgen observed a fluorescent glow of crystals on a table near his tube. The tube that Röntgen was working with consisted of a glass envelope (bulb) with positive and negative electrodes encapsulated in it. The air in the tube was evacuated, and when a high voltage was applied, the tube produced a fluorescent glow. Röntgen shielded the tube with heavy black paper, and discovered a green colored fluorescent light generated by material located a few feet away from the tube. Röntgen also discovered that the ray could pass through the tissue of humans, but not bones and metal objects. One of Röntgen's first experiments late in 1895 was a film of the hand of his wife, Bertha.

Within a month after the announcement of the discovery, several medical radiographs had been made in Europe and the United States, which were used by surgeons to guide them in their work. In June 1896, only 6 months after Röntgen announced his discovery, X-rays were being used by battlefield physicians to locate bullets in wounded soldiers.

In 1922, industrial radiography took another step forward with the advent of the 200 000–volt X-ray tube that allowed radiographs of thick steel parts to be produced in a reasonable amount of time. In 1931, General Electric Company developed 1 000 000 volt X-ray generators, providing an effective tool for industrial radiography. That same year, the American Society of Mechanical Engineers (ASME) permitted X-ray approval of fusion welded pressure vessels that further opened the door to industrial acceptance and use [1].

1.2. Related studies

During the preparation of this work, an analysis was performed on a range of online solutions related to X-ray image interpretation. This review identified several systems that could share features with the proposed X-ray image interpretation system. However, none of these existing solutions offer a full approach. Some systems focus on models with high precision for detecting bone fractures, others provide a wide range of body parts for analysis, and some have a well-designed interface and application. Below, two examples of related products are presented to show the functionalities that could be implemented to make the planned system better than the ones mentioned.

The first example is BoneView, a clinical AI application developed by Gleamer.ai [7]. It efficiently detects fractures, dislocations, and other bone anomalies. Figures 1.1 and 1.2 present

snapshots of this web application. However, it lacks some of the functionalities, such as the ability for patients to connect with doctors. There is no integrated medical system that allows users to create accounts, schedule appointments, or initiate video calls or messages. Although doctors can sometimes correct AI predictions, these features and functionalities are absent in this application. The planned X-ray image interpretation system in our work aims to address these gaps by implementing such features.

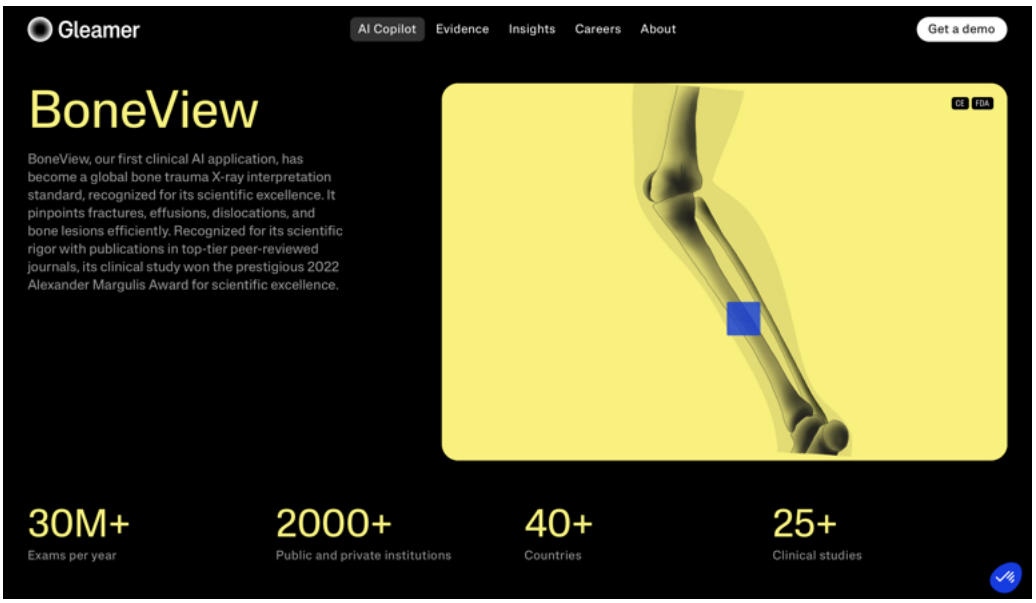

Figure 1.1: Homepage of BoneView, showing its role as a clinical AI solution for bone trauma X-ray interpretation.

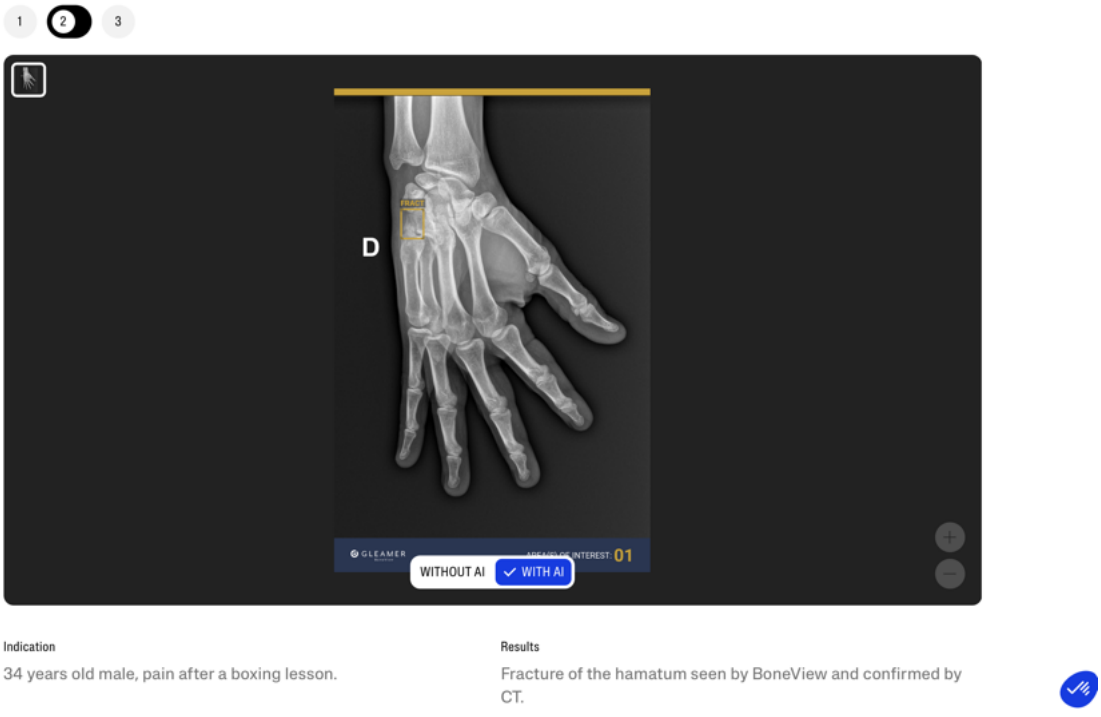

Figure 1.2: Example of BoneView detecting a fracture in an X-ray image using AI.

The second example is a bone fracture detection application developed as a final project, stored on GitHub [8]. Figures 1.3 and 1.4 show snapshots of this application. While it includes functionality to detect body parts and then identify fractures, it has limited features and application options. The system proposed in our thesis aims to improve upon it by offering a more functional interface, better security solutions, accessibility for patients and doctors, and detailed descriptions of detected fractures, generated by AI or verified by medical professionals.

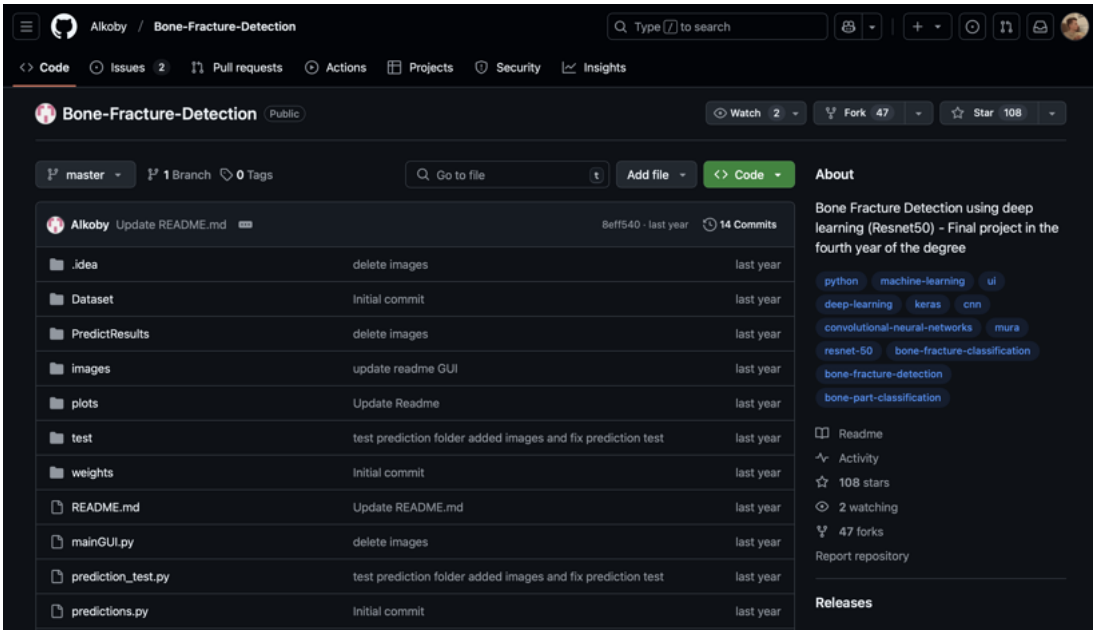

Figure 1.3: Main page of the GitHub repository for the Bone Fracture Detection application.

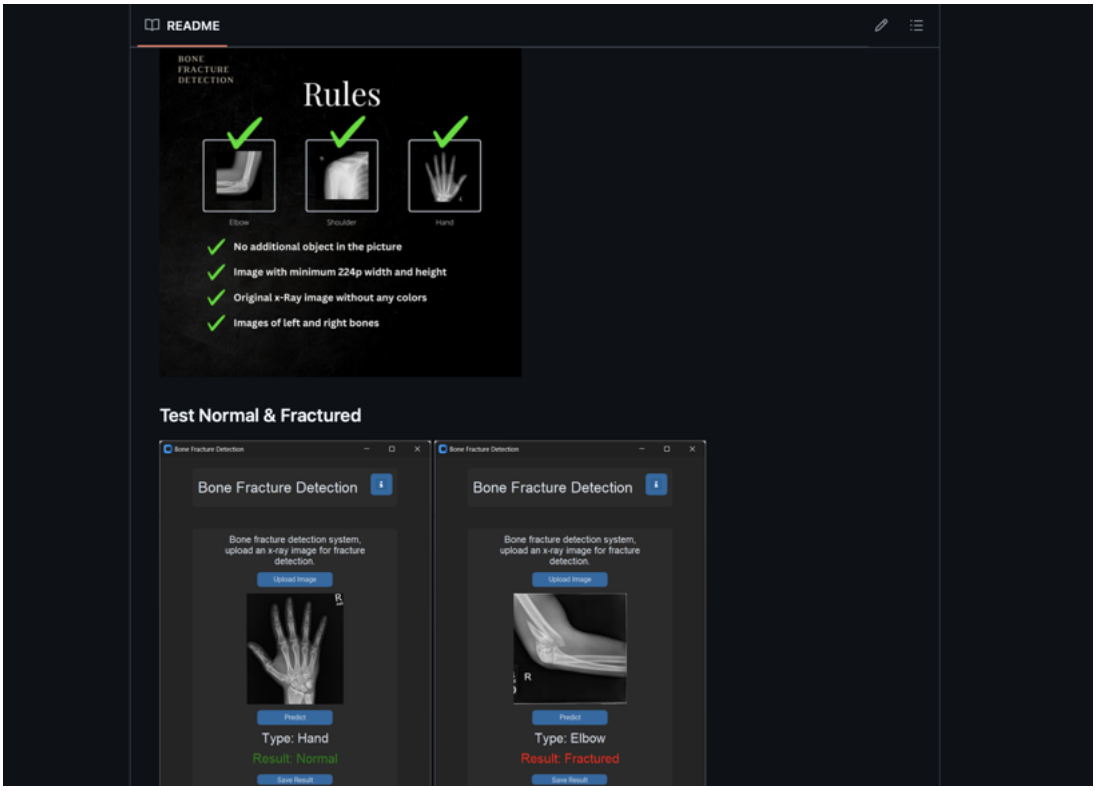

Figure 1.4: Screenshots from the Bone Fracture Detection application interface with examples of the detection system classifying X-ray images as normal or fractured.

1.3. Product vision and objective of the work

1.3.1. Application description and vision

The medical image analysis application is built to simplify the communication and analysis of medical data for patients and doctors, while keeping the data safe and meeting all the required regulations. The application is structured into frontend and backend components, leveraging React.js for the frontend and Java Spring Boot for the backend.

a. Frontend development

The frontend of the medical image analysis web application will be developed using modern web technologies such as HTML, CSS, and JavaScript, with a framework like React.js for dynamic UI rendering. The primary functionalities and components of the frontend include:

- **User Authentication:** To access the application, users need to log in as either patients or doctors to access their dashboards.
- **Patient Dashboard:** Patients logging into the application will be directed to a personalized dashboard displaying their medical information, upcoming appointments, and a secure chat and video call interface for communication with their doctors. The dashboard will be designed to be user-friendly and informative, providing easy access to relevant features.
- **Doctor Dashboard:** Doctors logging into the application will have access to a dedicated dashboard presenting them with lists of patients, medical records, and tools for image analysis. The dashboard will facilitate efficient management of patient data, appointment scheduling, and communication with patients through the built-in chat and video call feature.
- **X-ray Image Viewer:** A specialized component within the application will allow doctors to view and analyze X-ray images, including bone fractures highlighted by the machine learning model.
- **Responsive Design:** The frontend will be designed with a responsive layout to ensure optimal user experience. The UI will feature intuitive user interactions, clear navigation menus, and informative tooltips to guide users through the application's functionalities efficiently.
- **Security Features:** The frontend will implement secure components and client-side data validation to enhance security and protect user data.
- **API Integration:** The frontend will interact with the backend through a RESTful API, enabling real-time data updates and providing an efficient user experience.

b. Backend development

The backend will be developed using Java with the Spring Boot framework, providing a robust and scalable foundation for embedding databases, building RESTful APIs, handling business logic.

- **Database Management:** The application will use an H2 embedded database for storing patient data, medical records, appointments, and X-ray images. The database schema is designed to maintain relationships between entities efficiently, binding the photos of patients to them, and them to their doctors.
- **RESTful API Endpoints:** The backend will expose RESTful endpoints to handle CRUD (Create, Read, Update, Delete) operations for different entities. Implemented JWT-based authentication mechanisms will secure API endpoints and manage user sessions securely.
- **User Authentication and Authorization:** The backend will manage user authentication, authorization, and role-based access control (RBAC) to ensure data privacy and security. Business logic will extend to handling appointment scheduling and notifications for patients and doctors.
- **Encryption:** Sensitive data such as patient information and medical records are to be encrypted at rest and in transit to maintain confidentiality, as states HIPAA (Health Insurance Portability and Accountability Act) regulation for healthcare applications.
- **External APIs Integration:** The application will use externally deployed machine learning models for X-ray image analysis. Chat and video communication features will also rely on external APIs.
- **Logging:** Logging mechanisms to capture application events, API requests/responses, and errors for troubleshooting and auditing purposes.
- **Monitoring:** Monitoring tools may be integrated to track application performance, database metrics, and system health.

1.3.2. Vision of chat consultation feature

The interface will enable doctors to consult patients via direct text chat or video call. The architecture of the text and video chat functionality is designed to ensure scalability, reliability, and security. It encompasses various components including:

- **Frontend Interface:** Developed using React.js, the frontend interface provides a responsive user experience for both patients and doctors. It includes components for initiating, receiving, and managing text and video calls. The chat will enable users to send simple messages as well as sending attachments.
- **Stream Chat React SDK Integration:** The integration of Stream Chat React SDK facilitates the implementation of real-time text chat functionality. This offers functionalities such as user authentication, and real-time updates.
- **Video Calling SDK Integration:** The React Video SDK is integrated into the frontend to enable video calls between patients and doctors. This SDK provides essential features such as video streaming, audio communication, and screen sharing. This involves handling user permissions for camera and microphone access, and implementing UI elements for video call management.
- **Video Calling API:** The Video Calling API serves as the bridge between the frontend interface and the backend services. It facilitates the establishment and management of

video calls, handling tasks such as call initiation, participant synchronization, and call termination. This involves handling API requests, managing call state transitions, and implementing error handling mechanisms.

1.3.3. Federated learning and data privacy protection

Federated machine learning is a variation of machine learning when multiple clients train a single global model [9]. Additionally, clients can train local models that are tuned to client data which might better suit clients needs than a global model [10].

Traditional machine learning uses a centralized model with centralized data storage and data privacy protection requires external measures. Federated machine learning stores client data locally and only sends model updates to the global model allowing the model to learn without accessing client data. The main risk and challenge in federated machine learning is to ensure model updates sent to server will not reveal details about client data.

1.4. Risk analysis

In this chapter, a risk analysis of various factors associated with the project was conducted. Here are the key elements to consider when developing the application:

1.4.1. Incorrect model selection and improper deep learning training

The main aspect of this work is the proper training of models. One of the real threats is the incorrect choice of model architecture for the intended purpose. This is undoubtedly a crucial and most important element of the system. Improper model training can also result from insufficient training data or improper optimization of hyperparameters. To prevent this, it is necessary to test different configurations and model parameters.

1.4.2. Limitations of training data

There is a significant risk associated with selecting appropriate datasets. They must be representative and diverse to avoid model overfitting and ensure good generalization. Initial searches for datasets based on publicly available internet sources indicate that careful selection of suitable X-ray images will be necessary. There are few large, high-quality datasets with the necessary labels for training models that perform detection tasks. A potential solution could be to limit the scope to specific body parts for which sufficiently large datasets are available, along with appropriate image transformations, which may enhance model performance through improved generalization.

1.4.3. Personal data protection

Collecting images and creating user accounts involves gathering personal data. There is a real risk of data leakage or breach of personal data, which should be secured using data encryption technologies. In the case of models, it may be necessary to implement federated learning, which allows for preserving user privacy and protecting their personal data.

1.4.4. Infrastructure limitations

Computational constraints related to training and deploying models may exceed the hardware capabilities of available devices. It is important to consider these factors for system scalability and reliability. The application should be resilient to failures, long response times, and overloads.

1.4.5. Integration of different technologies and functionalities into one system

The process of integrating various components of the application can be complex. It is important to consider the selection of appropriate technologies that can be integrated seamlessly together. Initial attempts at developing the application have shown the value of using libraries that have support or the capability to integrate with other application components. It is also important to properly select a database system and integrate it with the web part and machine learning models. To minimize risk, it is advisable to regularly test the deployment of individual application components.

1.4.6. Potential lack of knowledge and experience in some technologies

There is a risk associated with a lack of familiarity with certain technologies that are crucial for the operation of this system. A significant challenge may arise from creating models that can deliver good results and developing a web application that can handle user queries. Inexperience in implementing advanced machine learning algorithms may lead to difficulties in effectively training and optimizing models, which in turn can affect the quality of X-ray image analysis. Furthermore, a lack of knowledge in web technologies and user interfaces may result in delays in implementing the web application and issues related to system performance and scalability. Integrating various functionalities, such as data management, presentation of analysis results, and user interaction, will also require skills in both back-end and front-end programming.

Chapter 2

Functional scope

2.1. User stories and characteristics

Applying an agile-based approach to developing applications allows for the creation of an actor model that outlines the application's overall functionality. This approach makes it easier to divide work stages into actors and stories, enabling effective task distribution within the project and the creation of subtasks. In addition, it helps identify users and define how they will interact with the application. In general, this approach simplifies and makes the development process more efficient in building functional applications.

2.1.1. User profiles

User profiles can be categorized into two types: Patients and Doctors.

- **Patients**

  Patients are individuals who access the application to manage their medical records, seek medical advice from doctors, and view their personal medical information, which includes past diagnoses and treatment plans.

- **Doctors**

  Doctors are healthcare professionals who utilize the application to manage patient care, access medical records, and communicate with patients.

2.1.2. User stories

- **Logging into the application**

  As a user, I want to be able to log into the application as either a patient or a doctor so that I can access all the relevant functions and data.

- **Browsing the patient dashboard**

  As a patient, I want to be able to see a personalized dashboard where I can find my medical information, upcoming appointments, and interface for communication with doctors via chat or video call.

- **Browsing the doctor dashboard**

  As a doctor, I want to have access to a dashboard that contains a list of my patients, their medical history, a view for analyzing X-ray images, and the ability to manage patient data and schedule appointments.

- **Management of patient data**

  As a doctor, I want to be able to manage patient data, add comments about detected fractures by AI, create reports from X-ray image analysis, and share results with patients.

- **Viewing X-ray images**

  As a doctor, I want to be able to analyze X-ray images with the machine learning tool.

  As a patient, I want to load my X-ray images to be analyzed with the help of machine learning model.

- **Application security**

  As a patient and doctor, I expect my data to be secure. I want the application to use data encryption and communication tools.

- **User authentication**

  As a patient and doctor, I require secure authentication to access relevant functions and data based on my role.

- **Integration of online consultations**

  As a patient, I want to communicate with doctors via text chat or video calls to receive medical advice.

- **Anonymization of X-ray images**

  As a patient, I would like to have the option to consent to anonymize my X-ray images for use in machine learning models.

- **Selection of body part and uploading X-ray image**

  As a user, I want to choose the relevant body part for examination, so that I can upload corresponding X-ray images to the system for interpretation.

2.2. Functional requirements

1. **Uploading photos** - Users should be able to upload X-ray images securely, ensuring that the system can process and analyze these images accurately for diagnostic purposes.

2. **Reviewing photos manually** - The application should provide tools for doctors to review uploaded X-ray images, including a section for adding comments and corrections.

3. **Real-time updates and notifications** - Users (both patients and doctors) should receive real-time updates and notifications regarding appointment reminders, new messages in the chat feature, and important updates related to their healthcare interactions within the application.

4. **Input of new medical data** - Doctors should have the ability to update patient medical data securely within the application, ensuring accurate and up-to-date information for effective patient care and treatment planning.

5. **External APIs integration** - The application must integrate with external APIs for features such as communication (chat, video), machine learning services (image analysis), and possibly external databases (medical records integration). API integration should be secure, reliable, and well-documented to ensure seamless interoperability and enhanced functionality within the application.

6. **Security** - The application must adhere to HIPAA security standards to ensure the confidentiality, integrity, and availability of patient health information (PHI). This includes implementing encryption for data at rest and in transit, access controls based on user roles, audit trails for monitoring data access and modifications, and regular security assessments and updates to safeguard PHI against unauthorized access or breaches.

7. **Text chat functionality** - The application allows users to initiate and participate in real-time text chats. This includes sending simple messages as well as attachments, with functionalities such as user authentication and real-time updates.

8. **Video call functionality** - The application enables patients and doctors to initiate, manage, and participate in video calls. This involves handling permissions for camera and microphone access, video streaming, audio communication, screen sharing, and UI elements for video call management.

9. **Machine learning model** - The application should include a separate machine learning model functionality to identify fractures in patients' bones.

10. **Machine learning metrics** - The machine learning model should incorporate metrics for evaluation.

11. **Highlighting results** - The application should also feature functionality to highlight and annotate the model's results.

2.3. Non-Functional requirements

1. **Efficiency** - Fast reactions with minimal delay where possible. Effective usage of system resources to handle multiple users.

2. **Usability** - Intuitive user-friendly interface with easy navigation.

3. **Reliability** - Stable and resilient, minimal number of errors, crashes. Minimal downtime. Mechanisms to ensure data accuracy and consistency. Reliable error handling.

4. **Validation** - Testing to ensure system's ability to handle expansive number of concurrent clients. Tests in high demand and adverse scenarios.

5. **Data protection** - Data protection with sufficient measures to ensure data security.

2.4. Users interaction and system functionalities

In the planning phase of system development, it is essential to visualize key interactions and functionalities using tools such as use case diagrams. These diagrams help clarify the roles of different users and the core functionalities of the system. Figure 2.1 presents a use case diagram that illustrates the main interactions between patients, doctors, and the application.

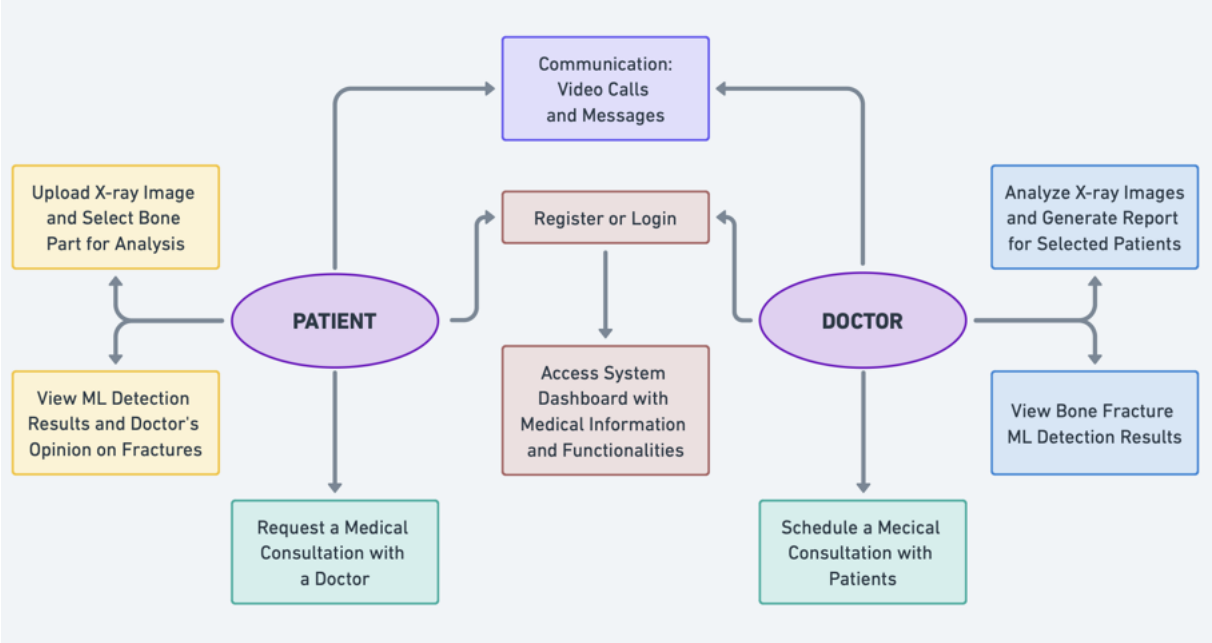

Figure 2.1: Use case diagram of the application's functionalities and user interactions.

The system must enable patients to interact with the application in several key ways. Patients should be able to upload X-ray images while selecting the specific bone part for analysis. They must also have the ability to view the results of machine learning detection and receive detailed opinions from doctors. Additionally, the application should allow patients to request consultations, communicate with doctors via video calls or messages, and access a personalized dashboard with their medical information.

For doctors, the system must provide view to analyze X-ray images and generate detailed reports for individual patients. It should also support reviewing machine learning detection results, scheduling consultations, and communicating directly with patients through video calls or messages. Furthermore, the application must include a dashboard that shows an overview of the relevant patient data.

Chapter 3

Selected realization aspects

This chapter provides a technical documentation of the project, describing its structure, components, and implementation.

3.1. Detection of wrist fractures

3.1.1. Dataset information

This dataset contains a large number of X-ray images related to the analysis of abnormalities in the wrist, named GRAZPEDWRI-DX. The dataset represents a solid collection of images ready for creating deep learning models aimed at detecting abnormalities. The images included in the dataset were gathered from 6091 patients at the University Hospital Graz between 2008 and 2018. Initially, the collected images were in DICOM (Digital Imaging and Communications in Medicine) format, a popular format in medicine that allows for seamless exchange between medical institutions [11]. Due to the complexity of this format and to facilitate analysis for researchers involved in machine learning, they were converted to a 16-bit grayscale PNG format. The dataset contains a total of 20 327 images in this format. This conversion allows for easier reading on computers and better support for many current libraries related to machine learning and general data and image analysis.

In addition to the images, the dataset contains annotations related to various abnormalities. Generally, there are several types of labels, such as:

- Fracture
- Periosteal reaction
- Pronator quadratus sign
- Soft tissue swelling
- Foreign body
- Bone anomaly
- Metal (foreign body)

For the purposes of generalizing the dataset within the overall system and application, it was decided that only fractures would be used as a ground truth, marked with bounding boxes as labels, while the rest of the elements and areas of the image would be treated as background.

This approach not only enables generalization but also allows for the implementation of a greater number of machine learning models, including simpler solutions. The labels are stored as bounding box coordinates in JSON files for each image individually. Example images of the given data set, along with annotations, are presented in Figure 3.1.

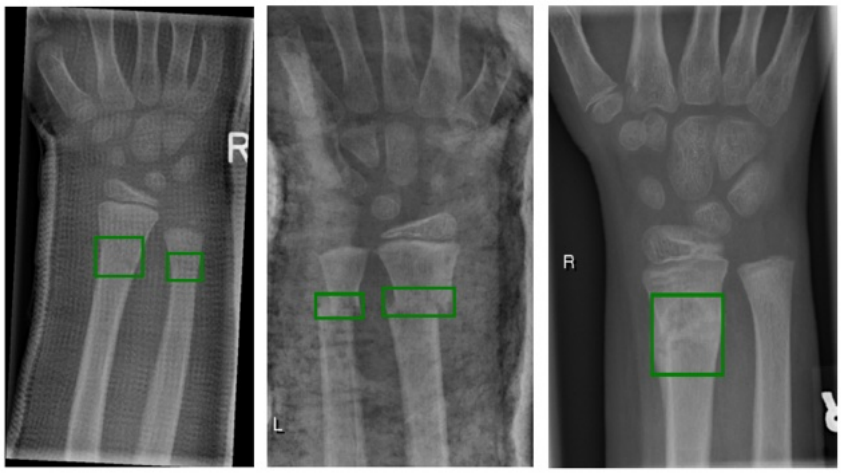

Figure 3.1: Examples of X-ray images with annotations indicating detected abnormalities.

A large number of annotations relates to bone fractures in this dataset collection. It guarantees that there is no significant decrease in the dataset's size. In the end, a total of 13 550 X-ray images featuring annotated bone fractures were employed for the thesis study.

The individuals on whom the radiographs were taken were between the ages of 2 months and 19 years old. This makes the dataset suitable for training a model to assist in interpreting X-ray images for children and adolescents. Interestingly, some of the images appear similar to these taken for average adult patients, which is why it was decided to include the wrist in the interpretation system. Figure 3.2 shows the age distribution of patients with detected fractures in the provided images.

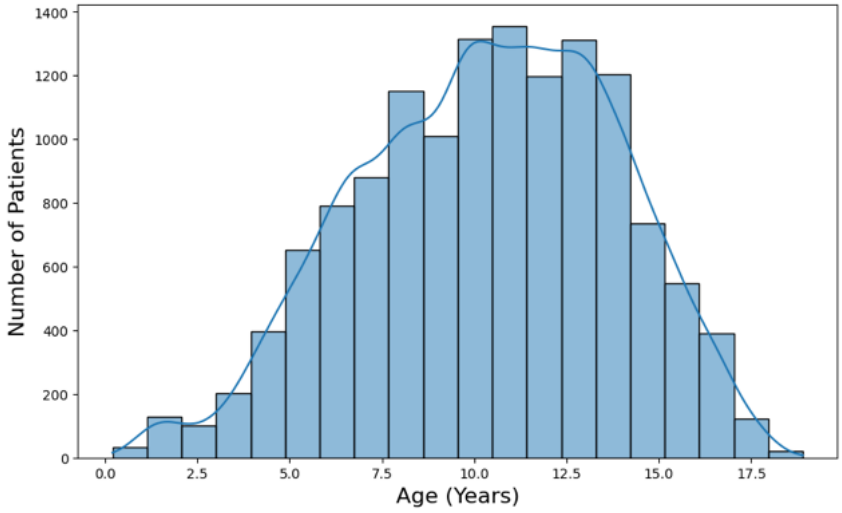

Figure 3.2: Age distribution of patients with detected fractures in wrist X-ray images.

Most of the patients were male, with a total of 8731, while there were 4819 female patients, as shown in Figure 3.3.

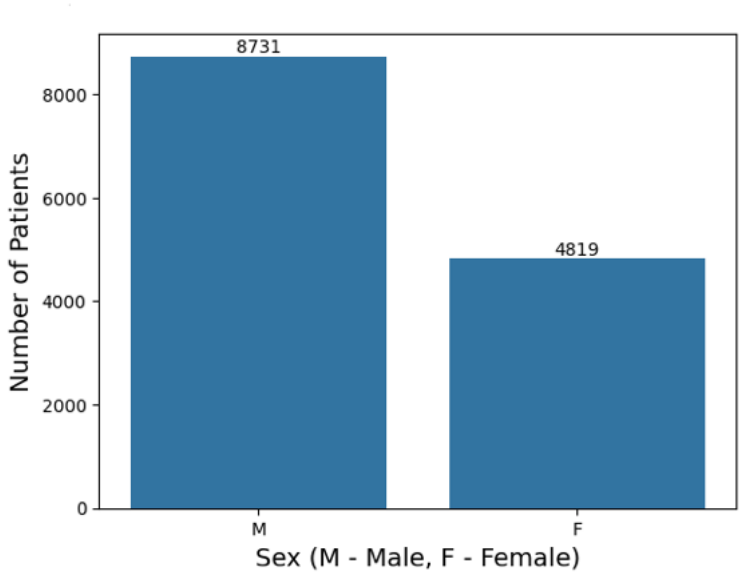

Figure 3.3: Number of patients according to sex in wrist X-ray images.

The advantage of this dataset is that the scans were performed in two different projections: lateral and posteroanterior (PA). In medical imaging, the term "posterior" refers to the back side of the body, in this case, the wrist, while "anterior" refers to the front side. The number of images taken in the PA projection was 6924, while there were 6553 in the lateral projection, as shown in Figure 3.4. This differentation in image projections helps facilitate training and prediction for the model, leading to better generalization and fewer errors in detection. Current deep learning models can easily learn to recognize different angles or positions of an object or body part.

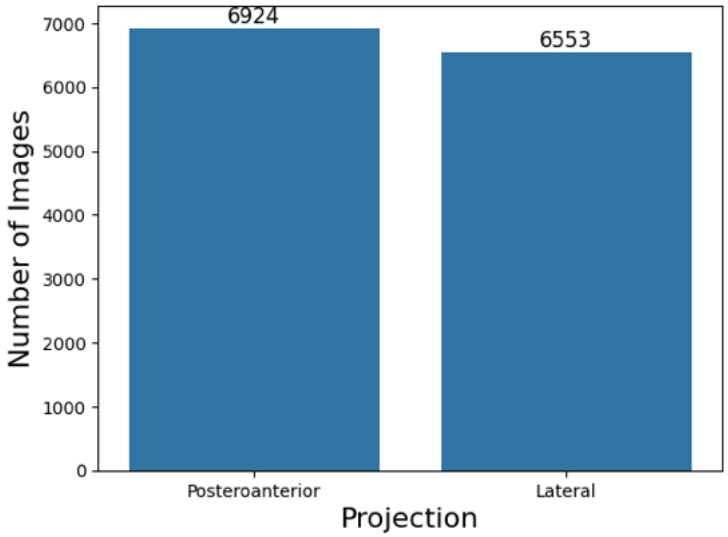

Figure 3.4: Projection types in wrist X-ray images.

A significant advantage of using this dataset for training neural networks is that it is very well-prepared, eliminating the need for complicated methods adjusting the images for the model, augmenting the dataset, or applying complex image transformations. It also represents a typical X-ray image that patients receive after undergoing an examination.

3.1.2. Faster R-CNN model

The Faster R-CNN model was chosen for its overall performance and the scores it achieves when used, making it an excellent baseline model for detecting bone fractures in X-ray images. This section describes the procedure for handling annotated data, the division of the dataset, the selection and creation of the model, as well as the training, validation, testing processes, and areas for improvement.

a. Data handling

To facilitate the separation of images for training and validation of the model, it was initially decided to divide the dataset into training, validation, and test sets. The annotations were also divided based on the image identifiers, which are stored in separate JSON files. To enhance reproducibility and speed up the notebook for analyzing and handling models, lists of file paths were created for both the images and labels, which are located in separate folders and share a common identifier. In cases such as image analysis, access through references is essential rather than storing entire objects in variables, unlike with tabular data. The list of references was divided into proportions: 70% of the bone fracture data for training (9485 images), a validation set comprising 20% (2710 images), and test images making up 10% of the original dataset (1355 images), as illustrated in Figure 3.5.

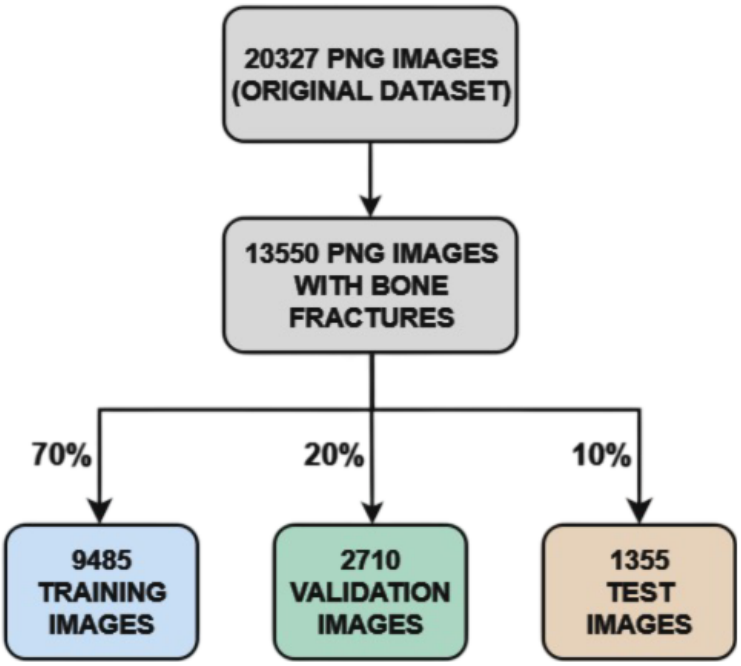

Figure 3.5: Proportional division of the dataset for bone fracture detection into training, validation, and test sets in a 70/20/10 ratio.

To ensure that images are properly loaded into the machine learning model and the necessary variables are set, a separate class named `FractureDataset` was implemented. This class is built on top of PyTorch's `Dataset` class, from which it inherits methods and can be extended to suit the specific dataset. This approach enables the model to efficiently load and process both the images and their corresponding annotations – in this case, the four coordinates of the bounding box corners $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$, and the fracture label. The pseudocode for this class is presented in Listing 3.1.

```
CLASS FractureDataset
    FUNCTION initialize(file_list, img_dir, ann_dir, transforms=None)
        STORE file_list, img_dir, ann_dir, transforms

    FUNCTION get_length()
        RETURN number of items in file_list

    FUNCTION get_item(index)
        LOAD image from img_dir
        LOAD annotations from ann_dir
        INITIALIZE empty lists for boxes and labels

        FOR each object in annotations
            IF object is a 'fracture'
                EXTRACT bounding box coordinates
                ENSURE coordinates are within image bounds
                ADD coordinates to boxes list
                ADD label for 'fracture' to labels list

        CONVERT boxes and labels to tensors
        CREATE target dictionary with boxes, labels, bounding box area,
   and iscrowd parameter

        IF transformations are provided THEN
            APPLY transformations to the image

        RETURN processed image and target data
```

Listing 3.1: Pseudocode for handling fracture wrist dataset based on [11].

Below, the image processing pipeline using the `FractureDataset` class is described in detail.

- **Loading the image**

  Firstly, the pipeline involves reading data and images from disk using OpenCV library, which reads the image in BGR format and then converts it to RGB standard format. Next, the image is normalized to a floating-point format where pixel values range from 0 to 1. This transformation helps the model converge more quickly during training.

- **Loading the annotation**

  Information about bounding boxes are read from files for each image and it is checked whether each box lies within the image boundaries. Each bounding box is represented by its corner points, which later define the Region of Interest (ROI) in the image where the model should focus its attention during training.

- **Specifying target output information**

  Since the objects operating on the defined pipeline must meet certain requirements regarding the returned data, a target dictionary is created that fulfills the interface requirements of PyTorch in machine learning models, including the implementation of Faster R-CNN from this library. The boxes, labels, and image index are stored as tensors. Additionally, the PyTorch Faster R-CNN model requires defining the "area" of image's labels, which is calculated as a product of the width and height of each bounding box, and an "iscrowd" parameter, which indicates whether the bounding box represents a crowd of objects or a

single object. However, in the case of the analyzed dataset, there are no instances where fractures may overlap, so the tensor is set to an array of zeros. Finally, the class returns the image transformed to tensor and its associated targets.

b. DataLoader utilization

The last step in handling the dataset is the use of the DataLoader, which is utilized as a PyTorch tool that facilitates data management during training and validation. Its main function is to break down large datasets into smaller batches, which are then passed to the model in each training iteration.

- **Batch size**

  The key parameter is "batch size", which helps optimize memory usage and calculation speed. Ideally, one would input the entire dataset into the model at once, but this is almost always impractical due to hardware limitations. Modern processing units still have constraints on the amount of memory that can be used for computing coefficients in neural networks during dataset passes. In this case, a batch size of 4 was set due to significant hardware limitations and potential costs associated with purchasing subscriptions for cloud processing machines. While a larger batch size would have allowed for faster training and more stable gradient updates, the chosen size of 4 still provided satisfactory performance during the training process.

- **Shuffling**

  Another parameter set specifically for the training dataset is "shuffle". Shuffling is a crucial step in training deep learning models because it ensures that the order of samples fed into the model is randomized in each epoch. This prevents the model from learning patterns based on the order of the data, leading to better generalization.

c. Faster R-CNN model overview

Before the development of Faster R-CNN, models such as R-CNN and Fast R-CNN were available. Fast R-CNN model introduced a new approach by processing the entire image in a single pass through convolutional layers and then applying Region of Interest (RoI) pooling to extract feature vectors for each proposal. This innovation led to improvements in speed compared to the original R-CNN model. However, Fast R-CNN still relied on external methods, such as Selective Search, for generating region proposals [12]. Selective Search combines image segmentation with an exhaustive search by examining every possible location in an image, grouping similar regions based on color, texture, and size, and refining them hierarchically [13]. This component was an area where further improvements could be made.

In contrast, Faster R-CNN introduced the Region Proposal Network (RPN), integrating this component directly into the network itself. This eliminated the need for separate approaches based on external region proposals [14]. The main idea behind the Faster R-CNN model is to integrate region proposals as a fast way to identify parts of an image that may contain the object of interest.

Faster R-CNN consists of several components that together form an object detection system. Key elements include its backbone architecture, Feature Pyramid Network (FPN), Region Proposal Network (RPN), and Region of Interest (RoI) Pooling. Figure 3.6 illustrates the structure of the Faster R-CNN model along with its components, which will be discussed further in this work.

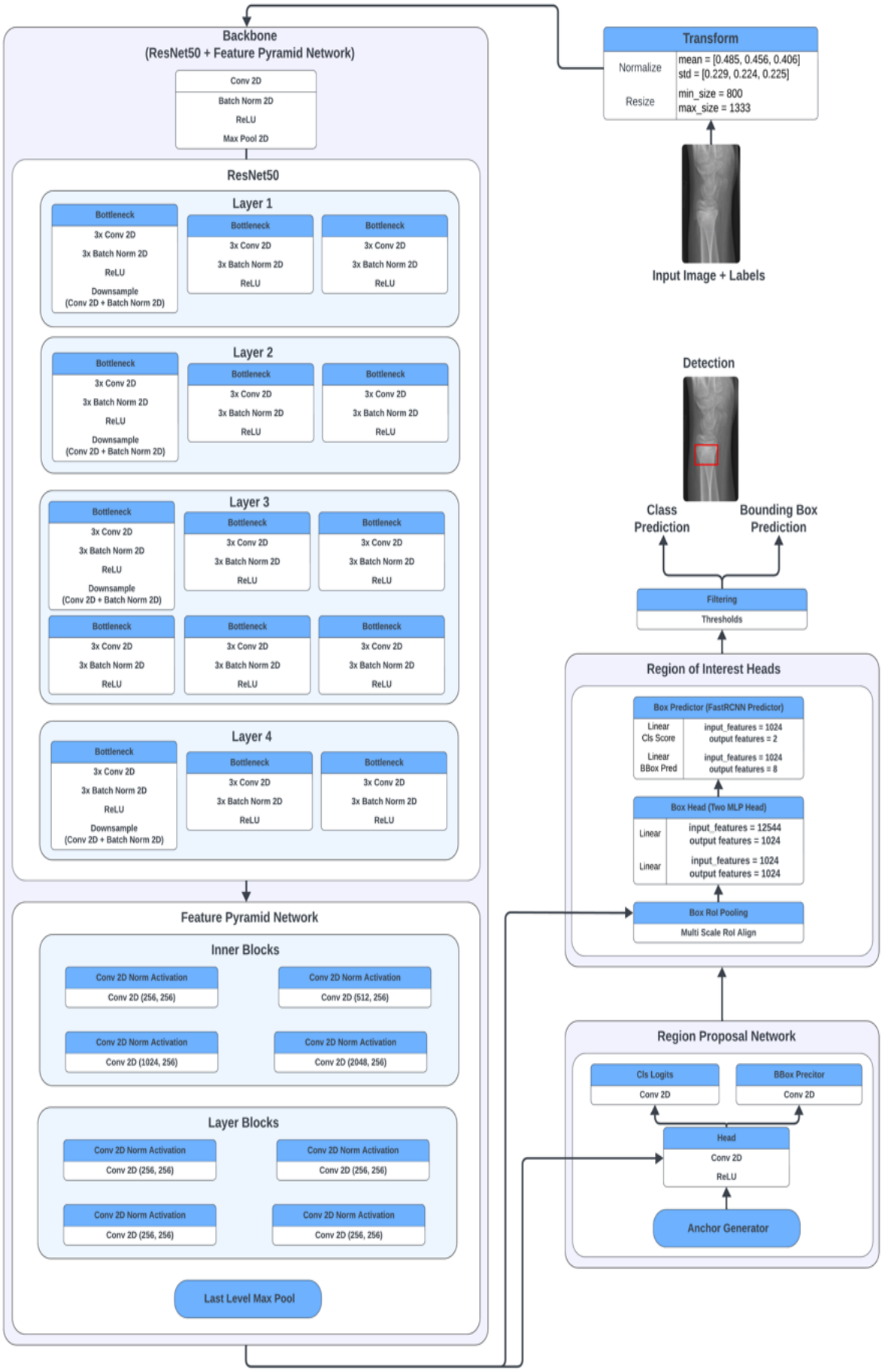

Figure 3.6: Schema of the Faster R-CNN model architecture, including its key components such as the backbone network, Region Proposal Network (RPN), Feature Pyramid Network (FPN), and Region of Interest (RoI) Pooling.

- **Backbone architecture**

  The backbone in machine learning models is primarily used for feature extraction, which helps the model learn patterns during training. For this project, a pre-trained ResNet-50 model was used, which is available in the PyTorch library and was trained on a large dataset called COCO (Common Objects in Context). This dataset contains over 80 object classes that the model can recognize in various contexts [15]. In this project, instead of using the original 80 classes, the number of classes was reduced to just two: 'fracture' and 'background.' This adjustment was achieved using transfer learning, where the pre-trained model weights were tuned for detecting fractures in X-ray images, which allows faster convergence with less training data.

  ResNet-50 (Residual Network) is composed of multiple layers of convolutions and bottleneck blocks. The architecture starts with a convolution layer that applies a 7x7 convolution with 64 filters to the input image. By using a stride of 2, this layer reduces the spatial dimensions. Next, batch normalization and ReLU activation are applied. A max-pooling layer then further reduces the size of the feature maps while preserving essential spatial information. The number of channels increases as you go deeper into the network.

  Each bottleneck block consists of three convolution layers: a 1x1 convolution that reduces the number of channels, a 3x3 convolution that operates on the reduced dimensions, and a final 1x1 convolution that increases the number of channels again (for example, from 64 to 256 or from 128 to 512).

  The purpose of the bottleneck structure is to allow the model to learn complex features with the help of residual connections between blocks, which help to reduce the vanishing gradient problem during training [16]. At the end of the ResNet-50 layers, the feature maps have fewer spatial dimensions but a much higher number of channels, ranging from 64 up to 2048.

  The Feature Pyramid Network (FPN) is built on top of ResNet-50 and is responsible for further feature extraction. In this stage, feature maps from different depths of ResNet are processed. This allows deeper layers with rich semantic features to be combined with high-resolution features from earlier layers [17]. This helps the network learn not only details but also overall structure, texture, and colors. At the output of the FPN, the feature maps from all layers are passed to the Region Proposal Network.

- **Region Proposal Network**

  The Region Proposal Network (RPN) was introduced in the Faster R-CNN model as a new method for generating potential object locations. It includes an Anchor Generator that proposes a set of anchor boxes with different sizes and aspect ratios (a total of 9 anchors: 3 sizes and 3 aspect ratios) for each spatial location [14]. The RPN also has a head that processes information from the Feature Pyramid Network, along with class logits and a Bounding Box Predictor that predicts the object class and the position of each anchor box.

- **RoI Pooling**

  The information is then passed through Region of Interest (RoI) Pooling, which extracts fixed-size feature maps for each proposal. RoI Pooling makes sure that all the feature maps going into the classifier are the same size, even if the original proposals are different. This allows the model to use two fully connected layers to send information to the Bounding Box Regressor and Classifier. Finally, different filters and thresholds are applied to fine-tune the model using the validation dataset.

d. Training process

The training process for this model and dataset was executed on the MacBook Pro M1 14" with a 16-core GPU using the MPS (Metal Performance Shaders) backend in PyTorch. The local environment for this task was Jupyter Notebook.

- **Number of epochs**

  The model was trained for 10 epochs. This number was set based on the expectation that the model would have enough steps to converge and minimize loss. The number of epochs was also monitored, and in the case of a sudden increase in validation loss over multiple epochs, an early stopping criterion would have been applied.

- **Optimizer**

  The optimizer used was SGD (Stochastic Gradient Descent with Momentum). This optimizer is a solid choice and is effective for object detection tasks. The learning rate used was 0.005, with a momentum of 0.9, and a weight decay of 0.0005. The momentum parameter helps the optimizer change directions and navigate across the gradient surfaces in the cost space more smoothly. Weight decay, on the other hand, prevents the model from overfitting by penalizing larger weights.

- **Scheduler**

  The StepLR scheduler was applied, which reduces the learning rate by a factor of 0.1 every 3 epochs. It allows the model to take larger steps in the starting phase of the training process and gradually adjusting the learning as the training progresses, allowing to get closer to the local minimum in the loss space.

- **Loss function**

  For this model, two types of loss functions were applied in the training and validation losses based on the article [14] and the official PyTorch library. These are the RPN losses and the RoI heads losses. The RPN losses consist of an objectness loss, which evaluates how well the network predicts whether a particular region contains an object. For this, binary cross-entropy was used. Additionally, this loss includes a bounding box regression loss, which checks the predictions of the box coordinates using smooth L1 loss. On the contrary, the RoI heads losses include a classification loss that uses cross-entropy and a bounding box regression loss in the RoI heads, also using smooth L1 loss [14].

In Figure 3.7, the training and validation cost functions are shown over 10 epochs. Both the training and validation losses decreased over the epochs, and the differences between them were relatively small, indicating that the model was learning well. From this visual assessment, we can conclude that there is no significant overfitting or underfitting present. In addition to the loss function results, it is also important to visually check the quality of the model's detections over the epochs. This is illustrated in Figures 3.8 to 3.17.

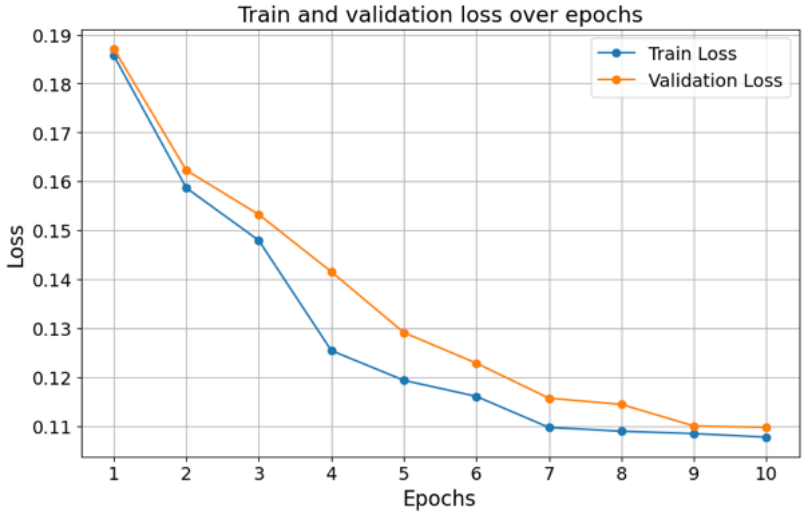

Figure 3.7: Training and validation losses over 10 epochs for the Faster R-CNN model.

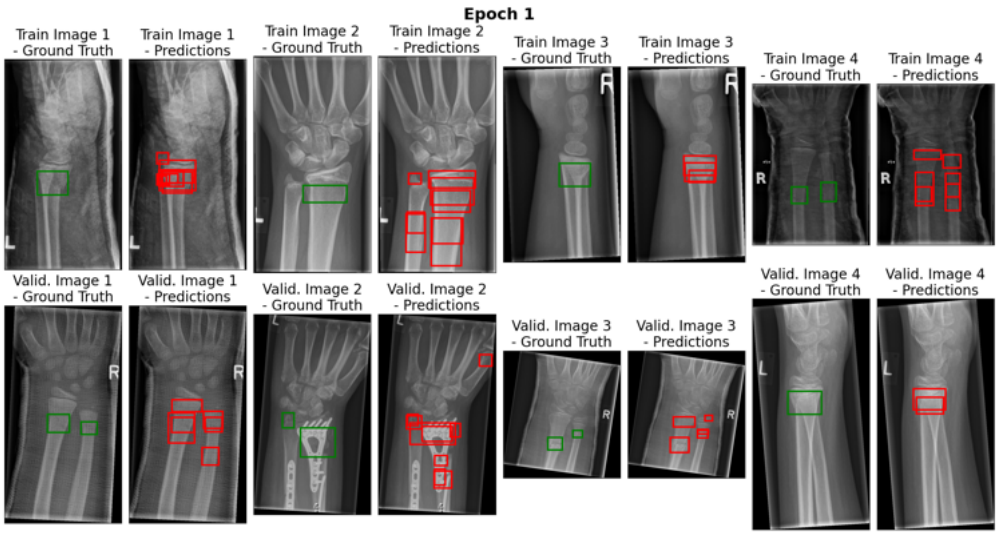

Figure 3.8: Training and validation examples from two random batches in epoch 1.

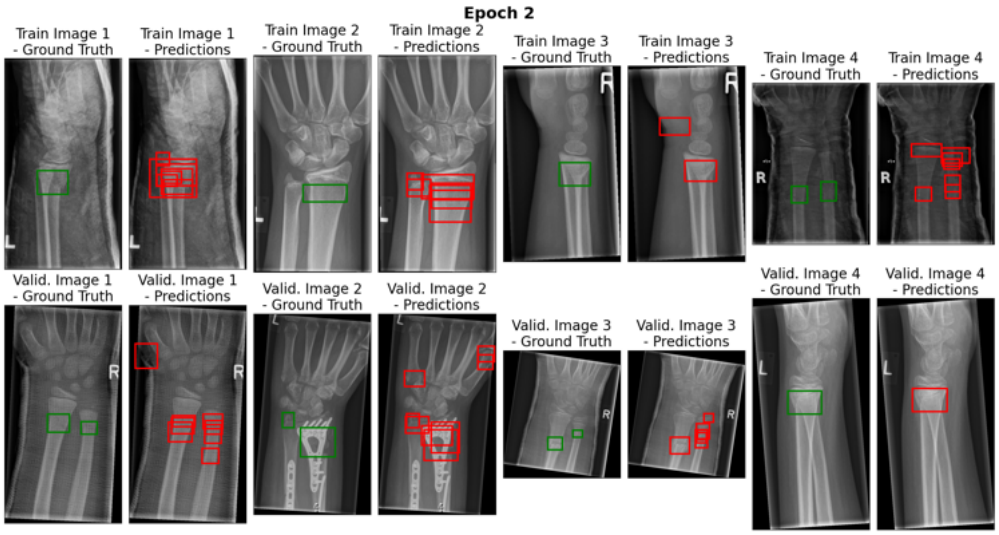

Figure 3.9: Training and validation examples from two random batches in epoch 2.

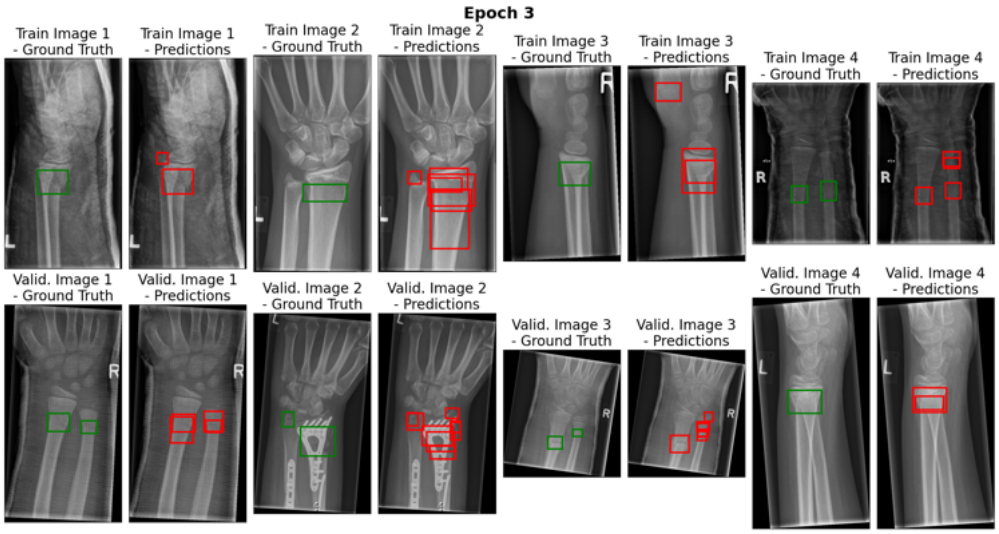

Figure 3.10: Training and validation examples from two random batches in epoch 3.

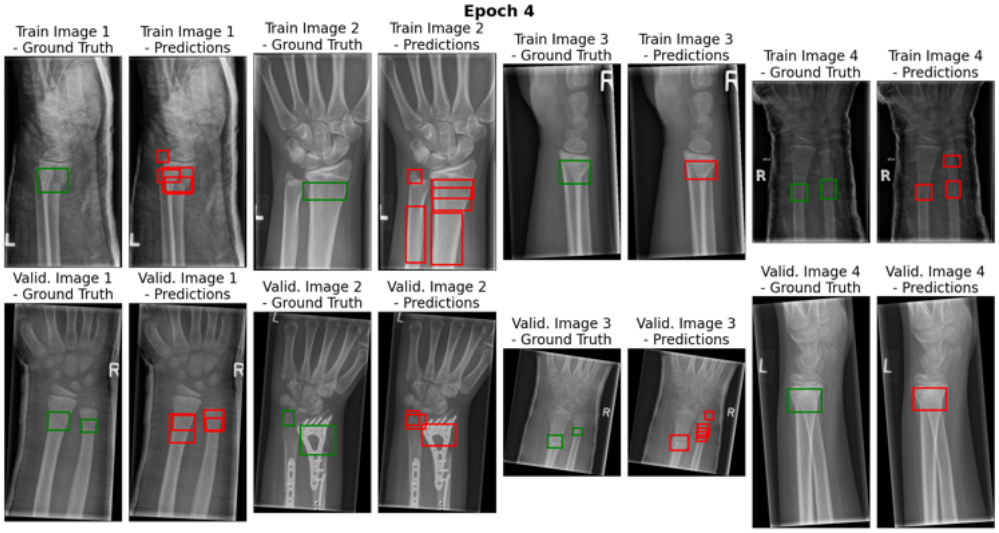

Figure 3.11: Training and validation examples from two random batches in epoch 4.

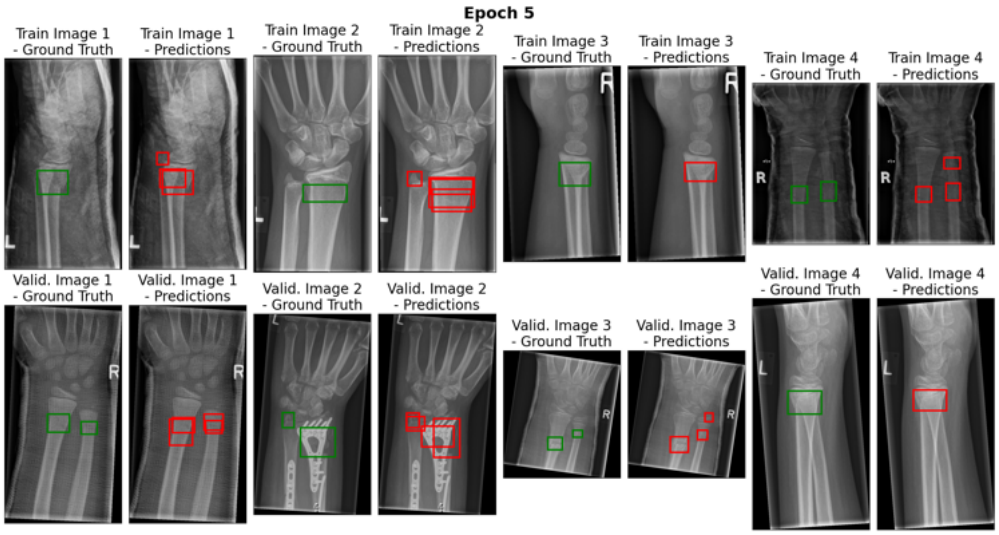

Figure 3.12: Training and validation examples from two random batches in epoch 5.

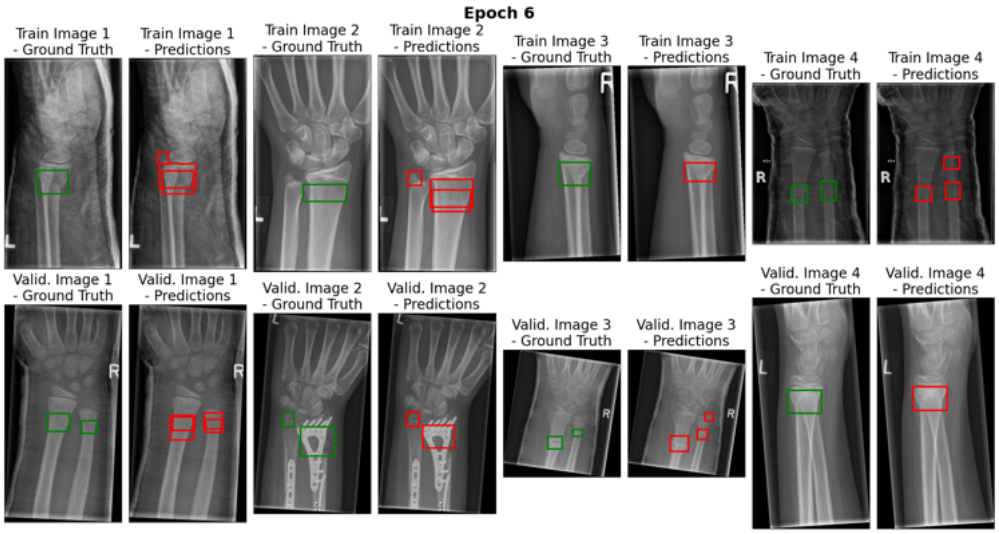

Figure 3.13: Training and validation examples from two random batches in epoch 6.

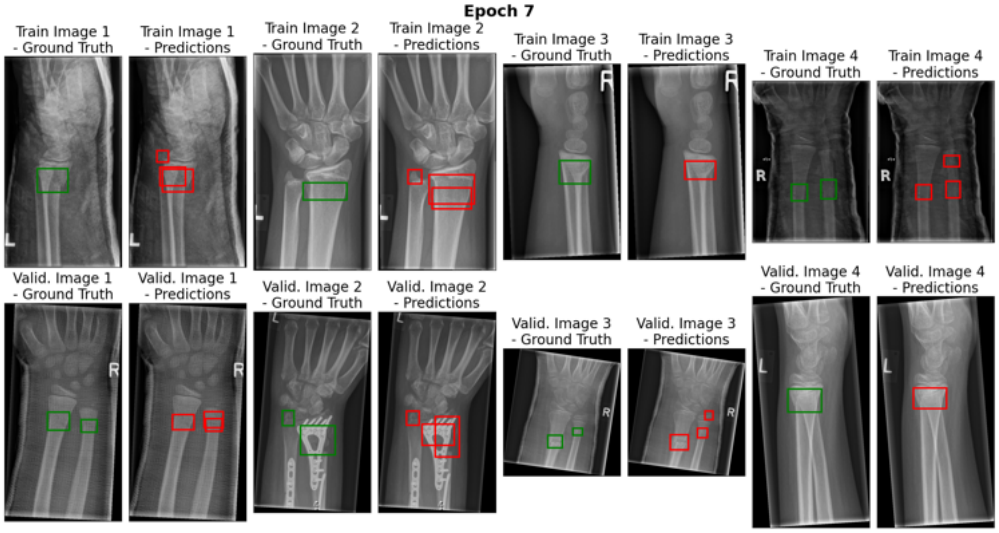

Figure 3.14: Training and validation examples from two random batches in epoch 7.

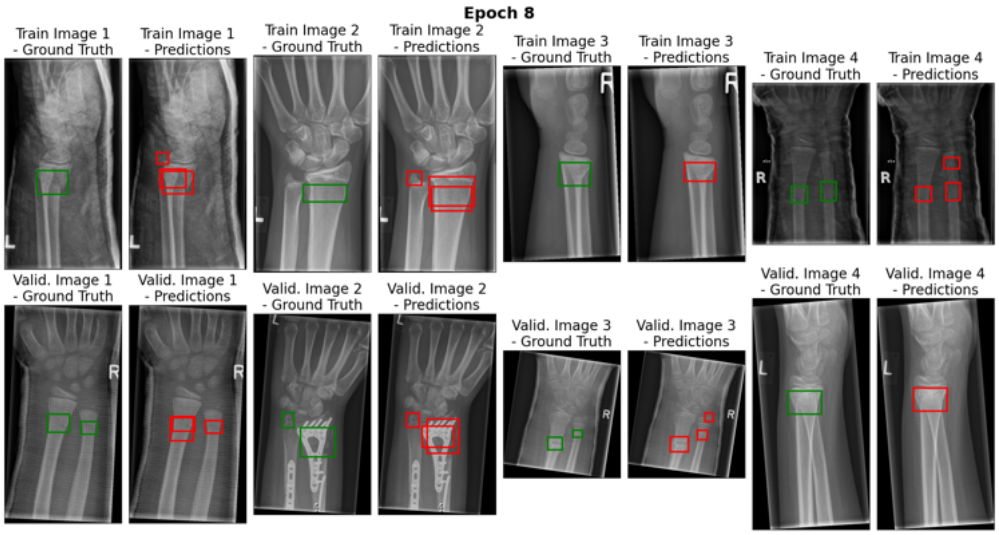

Figure 3.15: Training and validation examples from two random batches in epoch 8.

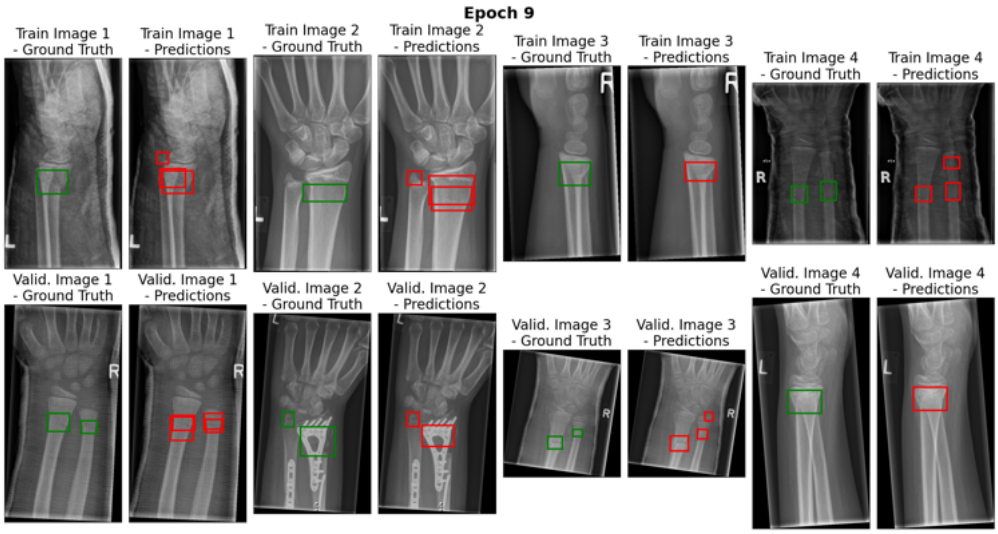

Figure 3.16: Training and validation examples from two random batches in epoch 9.

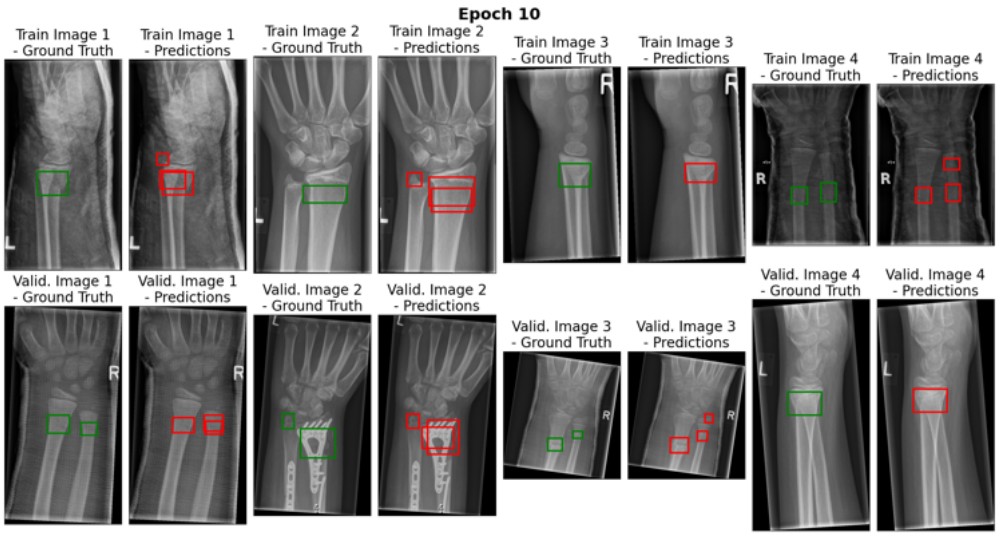

Figure 3.17: Training and validation examples from two random batches in epoch 10.

There is a significant improvement in detection from the 1st to the 6th epoch for both types of the example images in the training set and the validation set. As training progressed, the model mainly adjusted the number and size of bounding boxes to fit potential regions of interest.

However, even in the final epochs, the model still makes mistakes in some detections or generates too many predictions, resulting in a high number of false positives. This occurs because the loss function still has a non-zero value, even in the last epoch. Faster R-CNN models tend to generate many overlapping bounding boxes with different confidence scores during predictions. These results are saved in the model's state, allowing for hyperparameter tuning, which can significantly improve detection results, as discussed in the next subsection.

e. Validation and hyperparameters tuning

To further validate the model and improve detection results, hyperparameter tuning was applied. Precision-Recall curves were used as validation metrics to visually assess the model's performance. These curves are combined from the number of true positives, false negatives, and false positives, which together determine the values of precision and recall, as shown in equations 3.1 and 3.2.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.1}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.2}$$

In order to evaluate the model's performance during hyperparameter tuning, precision and recall are calculated and sorted based on the model's confidence scores for the predicted bounding boxes, in descending order from all predictions across the entire dataset. By lowering the confidence thresholds and calculating precision and recall for each threshold, a precision-recall (PR) curve is created. This curve helps to understand how well the model balances false positives and false negatives. However, the PR curve can sometimes appear in the shape of a "tooth" due to fluctuations in precision as the confidence threshold changes. Interpolated precision-recall curves are often used instead. In this version, the precision at each recall level is replaced with the highest precision observed at any later recall level. This creates a 'step' pattern in the curve, ensuring that precision either increases or stays the same as recall increases. Both curves are shown in the example Figure 3.18. For the purposes of further analysis, the interpolated PR curve will be used and referred as the PR curve.

Furthermore, the PR curves can be aggregated into a single value that summarizes the model's performance by representing the average precision for different recall values. This metric is known as mAP (mean Average Precision), which is the area under the precision-recall curve. The higher the mAP value, the better the model is considered to be. Therefore, this metric will be used in hyperparameter tuning as a measure of model evaluation.

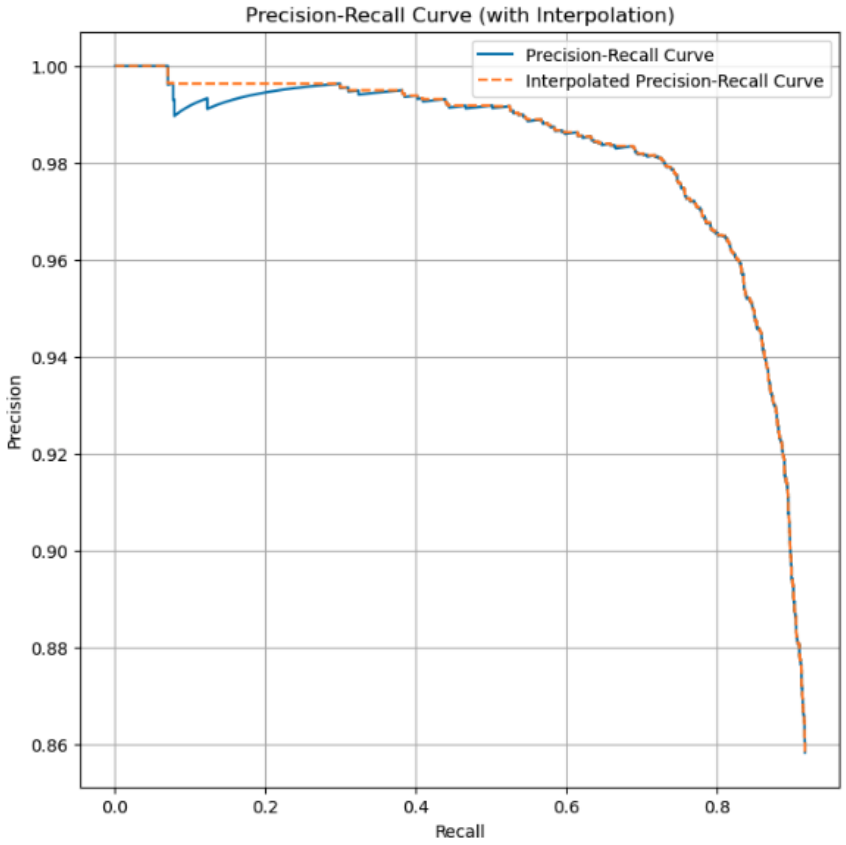

Figure 3.18: Example Precision-Recall curve with interpolation for the model without hyperparameter tuning.

For hyperparameter tuning, three model parameters were used:

- **Intersection over Union (IoU) threshold**

  Intersection over Union (IoU) is a metric used in object detection tasks to evaluate how well two bounding boxes overlap. It can be used as a threshold to determine when we classify a prediction as a true positive and when as a false positive. IoU is calculated by dividing the area of overlap between the predicted and ground truth boxes by the area of their union, resulting in a value between 0 and 1, where 1 indicates a perfect match [18]. The general formula is shown in equation 3.5.

$$\text{IoU} = \frac{(\text{Ground Truth Box} \cap \text{Prediction Box})}{(\text{Ground Truth Box} \cup \text{Prediction Box})} \tag{3.3}$$

  Figure 3.19 illustrates the overlap between the ground truth (green box) and the prediction (red box), along with their corresponding IoU values.

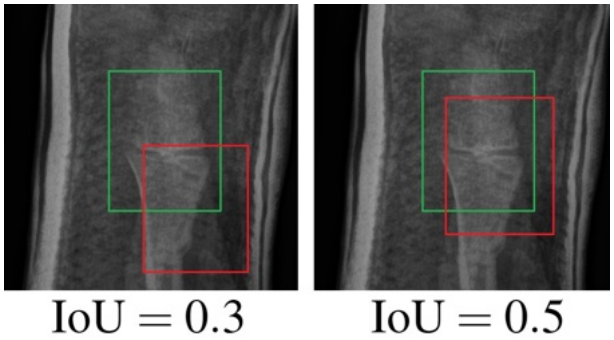

Figure 3.19: Example of Intersection over Union (IoU) based on the Prediction Box (red box) and Ground Truth Box (green box), along with their corresponding IoU values.

- **Confidence threshold**

  The confidence threshold is a measure used to determine the likelihood that an object belongs to a specific class, in this case, whether it is classified as "fracture" or not. When the threshold is set at a certain value, all confidence scores greater than this threshold are considered as model predictions.

- **Non-maximum suppression (NMS) threshold**

  Non-maximum suppression (NMS) is a technique used to eliminate overlapping bounding boxes. It is helpful when there are multiple overlapping predictions for the same object. In such cases, NMS selects the bounding box with the highest confidence score and does not consider the other boxes with an IoU overlap greater than a certain value as valid predictions. A typical value for this threshold is an IoU of 0.5.

For hyperparameter tuning, the following values were selected: confidence thresholds of 0.3, 0.5, and 0.7; NMS thresholds of 0.3, 0.4, and 0.5; and IoU thresholds of 0.5, 0.6, and 0.7. The tuning results are presented in Figures 3.20, 3.21, and 3.22.

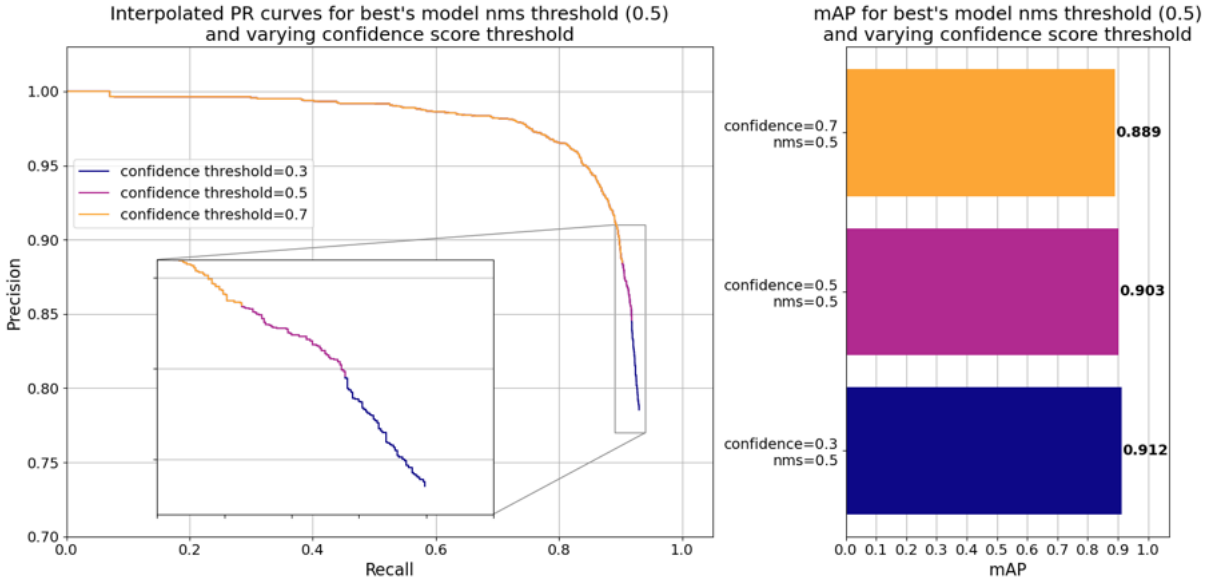

Figure 3.20: Precision-Recall curves showing three different models of Faster R-CNN for varying confidence thresholds (left) with a constant NMS threshold of 0.5. The bar charts on the right display the corresponding mean Average Precision (mAP) values for these models.

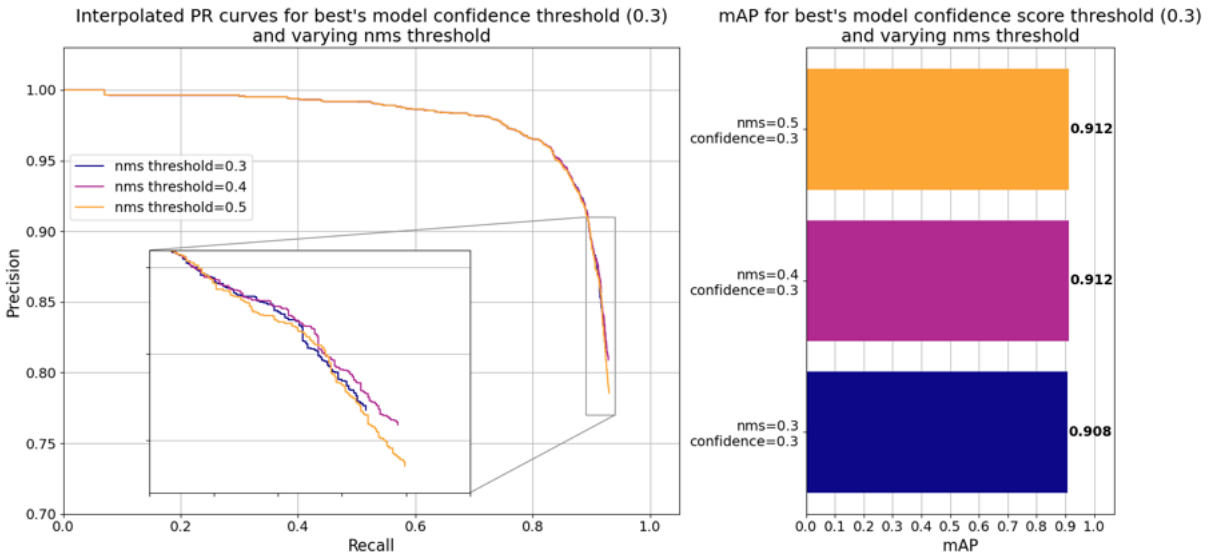

Figure 3.21: Precision-Recall curves showing three different models of Faster R-CNN for varying NMS thresholds (left) with a constant confidence threshold of 0.3. The bar charts on the right display the corresponding mean Average Precision (mAP) values for these models.

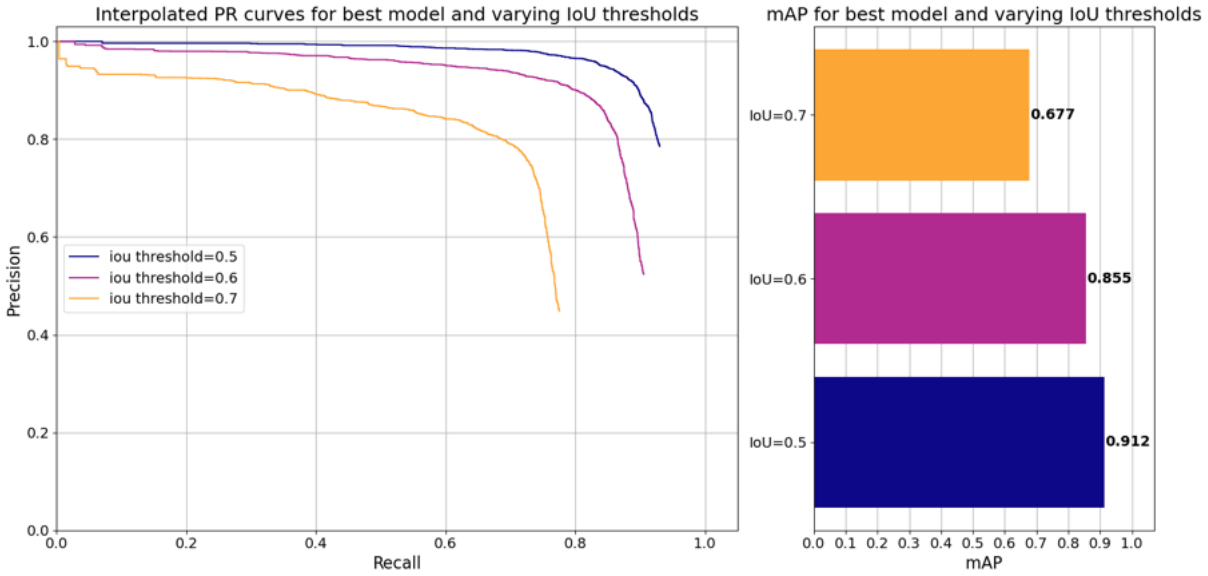

Figure 3.22: Precision-Recall curves showing three different models of Faster R-CNN for varying IoU thresholds (left) with a constant confidence threshold of 0.3 and NMS threshold of 0.5. The bar charts on the right display the corresponding mean Average Precision (mAP) values for these models.

f. Results and detection examples

Based on the validation results for various hyperparameters, the model with a confidence score threshold of 0.3 and an NMS threshold of 0.5 was selected as the final model for detecting fractures in the wrist parts of the patient body. This model achieved a high mAP score, a key metric in evaluating object detection models, and demonstrated very good precision and recall across the entire range of confidence threshold values. The balance achieved by this configuration ensures high accuracy in detecting fractures while reducing missed detections.

The model also performed well in detecting fractures on test set images, providing good results with its validation performance. Figures 3.23 and 3.24 present example batches of test data with detection results. In these examples, the bounding boxes generated by the model align well with the annotated ground truth. While the model occasionally over-detects fractures, this behavior is rare and occurs mainly in challenging cases where the X-ray images contain hard to detect regions. It is important to note that achieving high precision and recall values simultaneously is a difficult task due to the trade-offs between these two metrics. However, the performance of the selected model demonstrates a good balance suitable for medical applications.

The tendency of the model to slightly over-detect in certain situations also reflects its effectiveness in handling differences in X-ray images. Factors such as differences in patient anatomy, image quality, and imaging angles can introduce noise and uncertainty, making fracture detection a challenging task. Despite these difficulties, the model adapts well and identifies critical regions of interest. Additionally, the model's performance reflects the benefits of its architecture, particularly the Faster R-CNN architecture, which combines region proposal and classification in a unified pipeline. The use of a pretrained backbone (ResNet-50) ensures that the model uses rich feature extraction capabilities.

Overall, the results demonstrate that the model is well-suited for deployment in the image interpretation system. While no model can guarantee perfect predictions in every scenario, the selected configuration achieves a practical and effective balance. It provides accurate fracture detection while keeping patient safety as a priority.

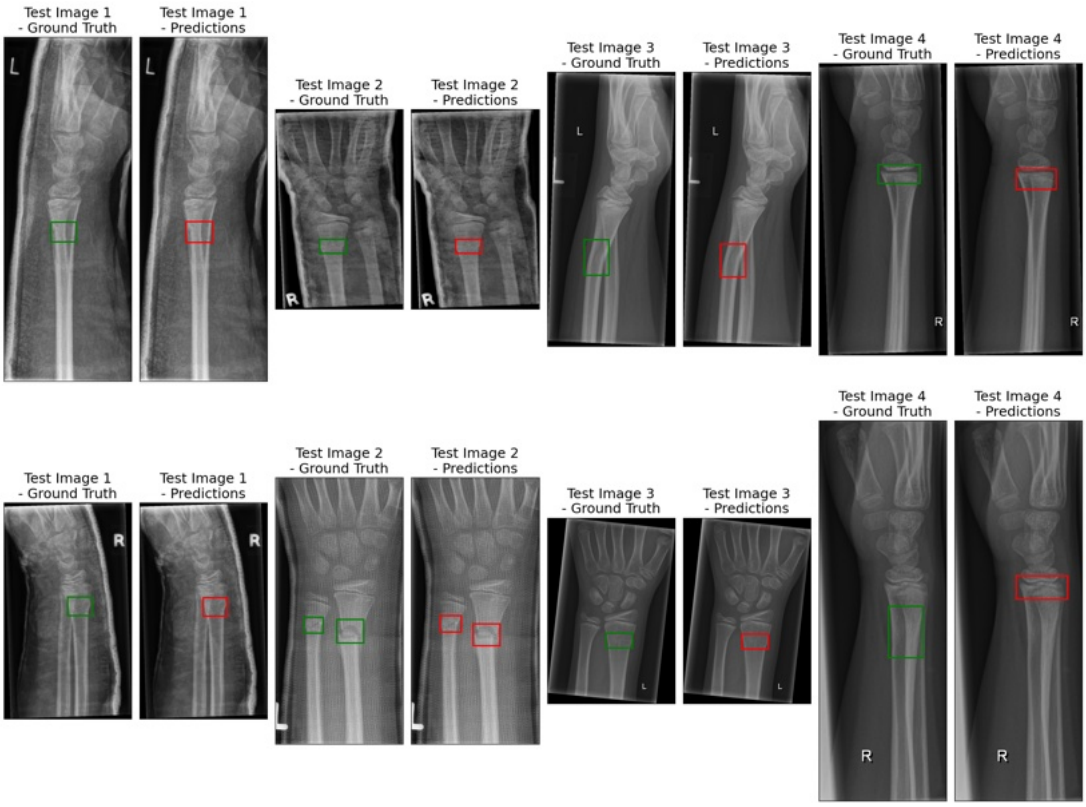

Figure 3.23: Fracture detection on test images from random batches using Faster R-CNN model.

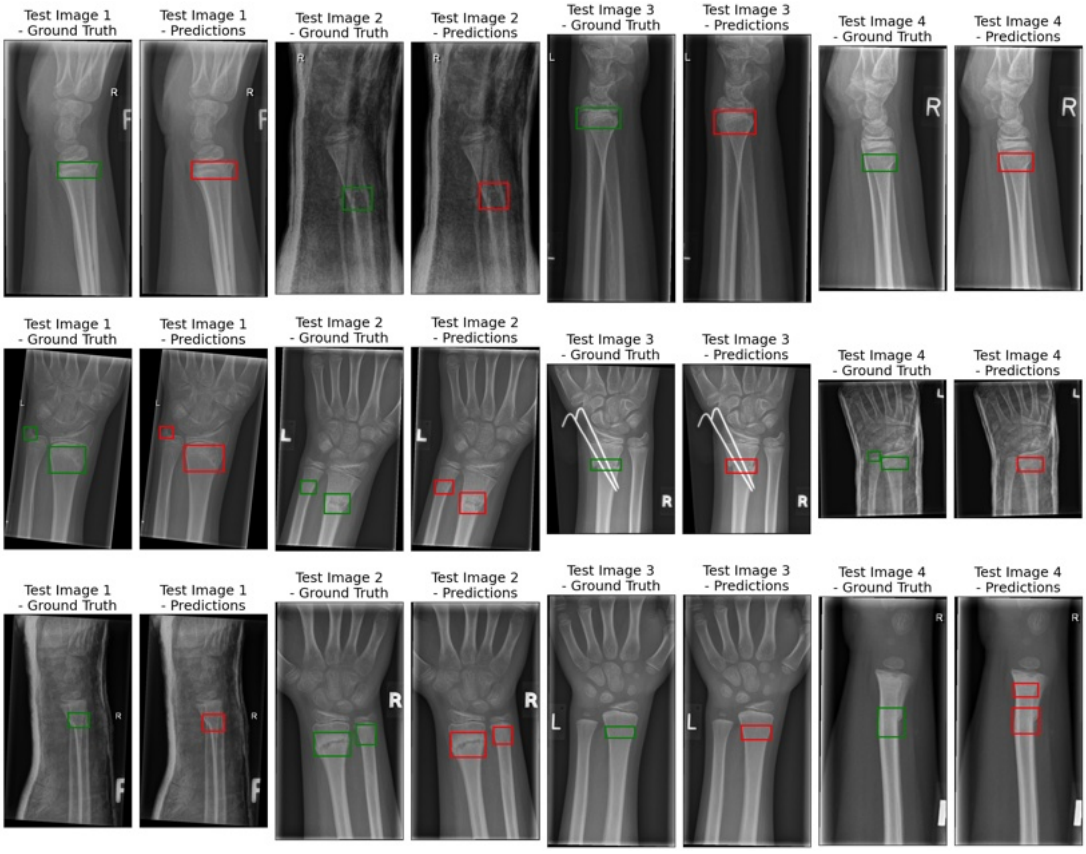

Figure 3.24: Fracture detection on test images from random batches using Faster R-CNN model.

g. Model limitations and improvement areas

- **Batch Size Limitations**

  Due to issues with the PyTorch library's implementation of the Faster R-CNN model on the MPS device backend, the training process was restricted to a batch size of 4. Larger batch sizes could have sped up the training process and allowed the model to view more images and variations at each step, helping it learn specific patterns in feature maps faster. This limitation likely affected the overall precision and speed of training.

- **Different Optimizers**

  The project used the SGD optimizer, but testing alternative methods such as Adam or RMSprop could potentially improve results. Adam, for example, adapts the learning rate for each parameter and can handle gradients better, leading to faster convergence. Also, exploring different optimization methods might result in better detection performance.

- **Different Optimizer Parameters**

  Hyperparameter tuning for different optimizer configurations, such as experimenting with various values of weight decay and learning rate, could improve the model further. Adjusting these parameters could help the model achieve better generalization and precision. Unfortunately, due to the limited computational power of the laptop, these experiments could not be performed.

- **Activation Functions**

  Testing different activation functions, such as ReLU, Leaky ReLU, or GELU, could improve the model's learning capabilities.

- **Backbone Architecture**

  The model used ResNet-50 as the backbone, but with access to a GPU with more memory and better performance, deeper backbones like ResNet-101 or ResNeXt could be tested. These architectures have more layers and extract features with greater detail, potentially improving the accuracy of the model.

- **New Images in Dataset**

  Collaborating with healthcare institutions or conducting anonymous surveys could help gather 'real-world' X-ray images. These images could be transformed into a format compatible with the current dataset and used alongside the existing data. This would allow the model to generalize better in real-world scenarios.

- **Ensemble Methods**

  Implementing ensemble methods by combining predictions from multiple models could potentially boost performance. For example, using different models and architectures, and aggregating their predictions might reduce false positives and negatives, leading to a more robust system overall.

3.1.3. YOLO model

The YOLO v9m model was selected for this project due to its popularity and ease of use in object detection tasks. It was decided to test whether the results reported in paper [19], which achieved a high mAP score of around 0.95 for the 'fracture' class, could be replicated. Additionally, YOLO v9m allows training an efficient model on a standard GPU, making it easy to use on personal hardware.

a. Data handling

The data from the wrist dataset was organized into specific folders to ensure efficient processing. To use the YOLO model from the Ultralytics library, the dataset structure had to follow the specific format required by the library. The dataset was divided into three main subsets: training, validation, and test, with each subset containing separate folders for images and their corresponding annotations. This folder structure helps maintain consistency for the training and evaluation processes. The folder structure is presented in Figure 3.25.

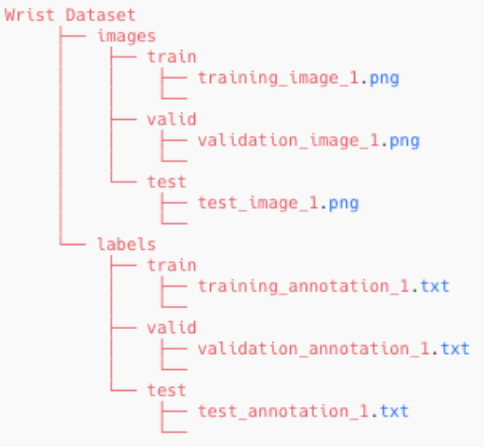

Figure 3.25: Folder structure for YOLOv9 model.

b. Training process

The YOLOv9 model was trained using the Ultralytics library on a MacBook Pro M1 14" with a 16-core GPU. The model was initialized with weights pretrained on the MS COCO dataset. The starting learning rate was set to 0.01, ensuring that the model could begin stable training with a small steps In the training process the Stochastic Gradient Descent (SGD) optimizer was used. This optimizer was configured with a weight decay of 0.001 and a momentum of 0.9 to prevent overfitting. Overall, the following key parameters were configured during training:

- Batch Size: 4
- Confidence Threshold: 0.001
- IoU Threshold: 0.7
- Maximum Detections per Image: 200
- Maximum Number of Workers: 8

The training process was stable over 100 epochs, with the loss decreasing consistently for both the training and validation datasets. The loss values are presented in Figure 3.26.

Figure 3.26: Training and validation losses over 100 epochs for the YOLO v9 model.

c. Results and detection examples

The YOLO model was validated using the same validation dataset as the Faster R-CNN model to maintain consistency in the results. The Precision-Recall curve and mAP score were used to evaluate the model's detection performance. Because of the large number of training epochs and limited hardware resources, the hyperparameters were not adjusted, and no additional tuning was performed. Despite this, the results were satisfying. The model achieved a mAP score of 0.93, showing that the selected parameters worked well for this task. The Precision-Recall curve is presented in Figure 3.27.

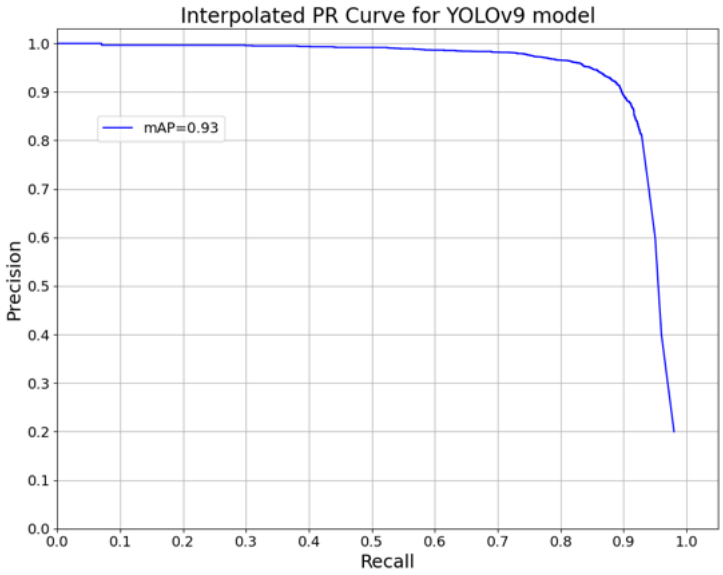

Figure 3.27: Interpolated precision-Recall curve for YOLO v9 model on the wrist dataset.

The YOLO v9 model performed similarly to the Faster R-CNN model in detecting fractures, as shown in Figures 3.28 and 3.29. In some cases, it improved the coordinates, sizes, and aspect ratios of the bounding boxes compared to the Faster R-CNN results. However, the model still produces a small number of false positives. In medical applications, this is acceptable and even desirable, as it ensures that potential fractures are not overlooked.

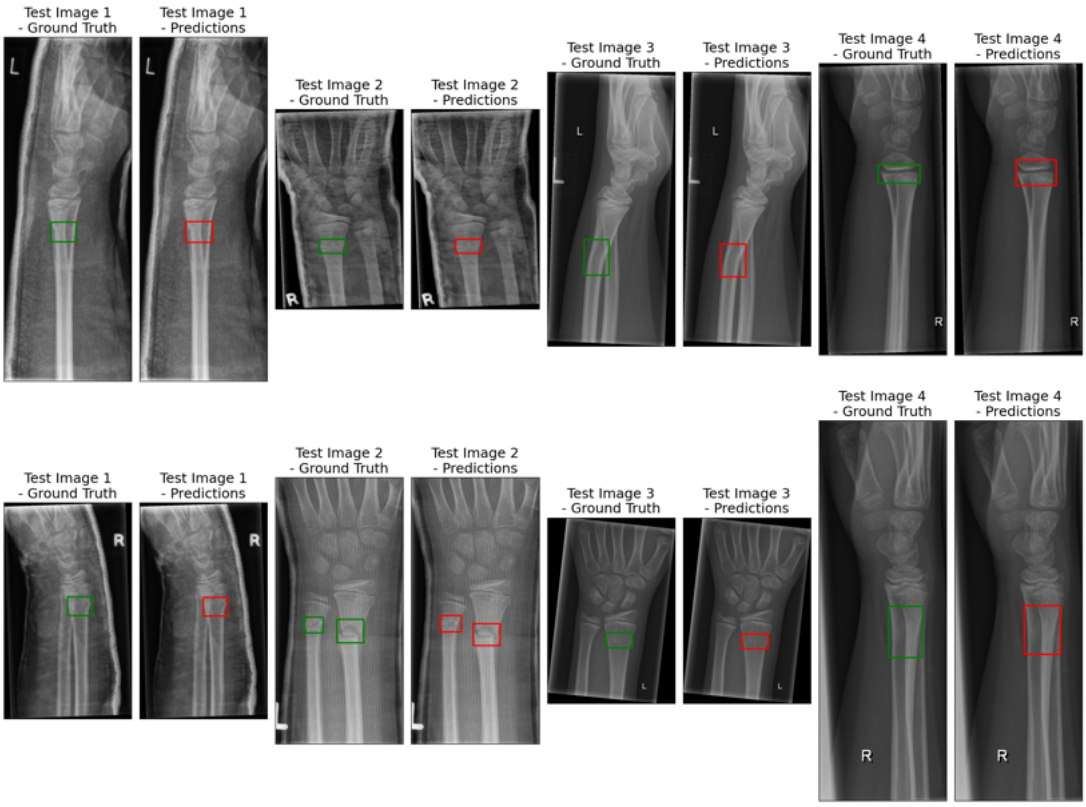

Figure 3.28: Fracture detection on test images from random batches using YOLO v9 model.

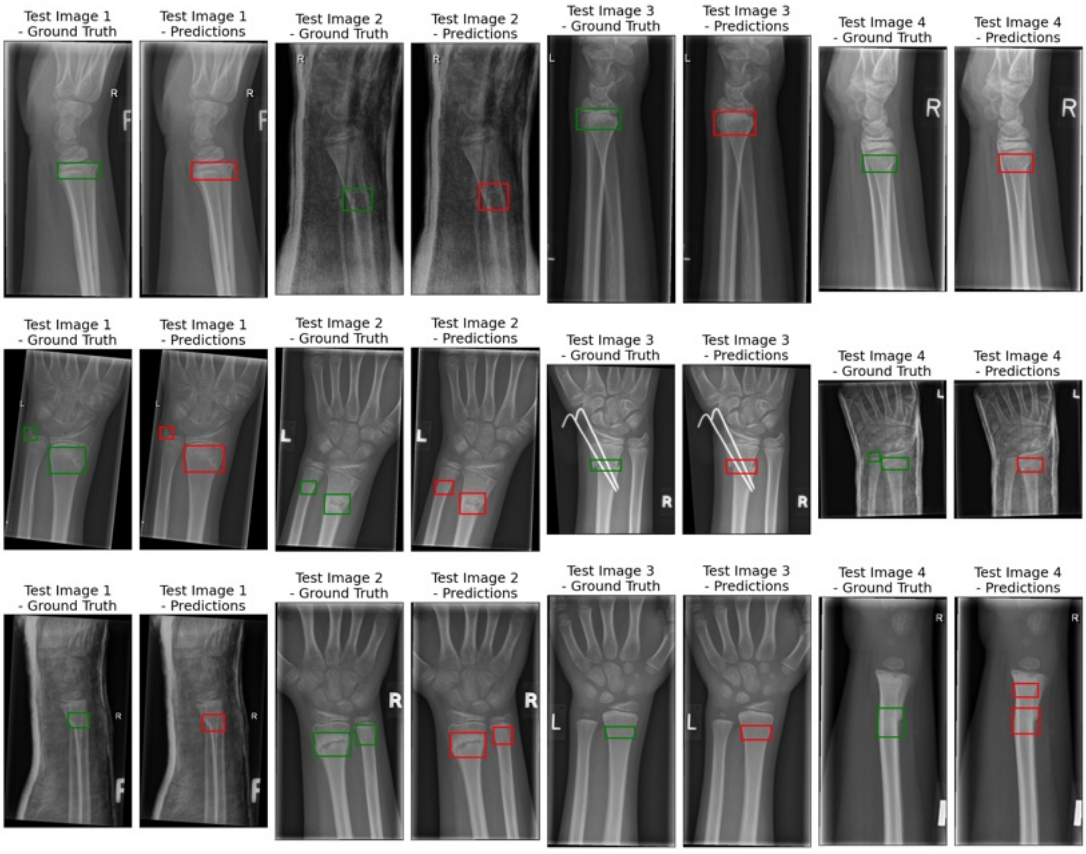

Figure 3.29: Fracture detection on test images from random batches using YOLO v9 model.

d. Model limitations and improvement areas

- **Batch Size**

  The batch size was limited to 4 due to hardware constraints. Larger batch sizes could improve training stability and speed.

- **False Positives**

  The model produces some false positives. While acceptable in medical cases, reducing these would improve precision.

- **Limited Tuning**

  Hyperparameters like learning rate and confidence thresholds were not thoroughly optimized due to time limits.

- **Using Larger YOLO Models**

  Testing larger YOLO versions, like YOLO v9 large, could boost performance.

3.1.4. Model integration in the X-ray image interpretation system

Two models for detecting bone fractures, Faster R-CNN and YOLO, were trained and prepared. This approach allows flexibility in the X-ray image interpretation system. If changes are needed in the application, the administrator can switch between models by simply replacing the model file in the source path of the application. This functionality is enabled by a Python script that creates endpoints. These endpoints handle sending X-ray images to the model and receiving the detection results from the model, which is saved in the folder structure path of the application and integrated with the system.

3.2. Federated learning

Federated learning (FL) involves training machine learning models in multiple separate locations with different local datasets. The key feature of federated learning is keeping user data on their device. Only model updates are sent to the server. This allows sensitive user data to remain private. To ensure data privacy, additional techniques may be required [20].

3.2.1. Vertical and horizontal federated learning

In vertical federated learning, different entities have data with the same set of users but different features. For instance, two companies operating in the same city may have a large number of overlapping users. Although both companies have the same entities, their feature space is different, as both companies may operate in different businesses. Vertical federated learning would allow them to train the model using data from both of these companies. In contrast, horizontal federated learning deals with a scenario in which different entities have unique data records in the same feature space. For example, each hospital may have similar databases but for different patients. Horizontal federated learning allows us to train the machine learning model using data from multiple sources while keeping sensitive patient data private. Our case is a horizontal federated learning - the feature space (X-ray images) is the same but comes from different users and patients [20].

3.2.2. Data heterogeneity

In horizontal federated learning data heterogeneity is an important aspect that impacts the learning process. Contrary to traditional machine learning, in federated learning data is distributed across different devices and might not be identically distributed. This creates several challenges such as bias towards over-represented data. Bias might lead to unequal performance in different data. Data heterogeneity might also complicate the privacy-preserving facet of the training process. It might also slow model converge or negatively impact model performance [21].

3.2.3. Pseudocode

The high-level pseudocode for federated learning is shown in Listing 3.2 [22].

```
global_model_initialization
for every training epoch
    connection to willing and chosen clients is established and the
    current global model send to them
    each of the federated clients executes their own training
    the model updates are send back to the server
    server aggregates received updates and updates global model
```

Listing 3.2: High-level pseudocode for federated learning

3.2.4. Dataset information

This dataset, named do1, contains a number of bone fracture images. The dataset does not contain any additional information. Bone fracture images are of high variety and are not limited to one body part. There are 1019 images in the dataset. They have been divided into 713 training images, 204 validation images, and 102 test images. [23].

This dataset contains a number of X-ray images of wrist bone fractures. This subset is named bone-fracture-yzkpo. The dataset originally contained 191 images. The images have been augmented to produce 455 images. Augmentations include horizontal and vertical flips and random rotations between -15° and +15°. The dataset with 455 images are divided into 399 training, 39 validation, and 17 test images [24].

Neither dataset contains any additional information. The downloaded datasets are in the YOLOv9 format which includes image and a number (typically 1 to 4) of bounding boxes with class label. In our case, the model only learns 2 classes: bone fractures and background class. Therefore, the bounding boxes in the dataset always have a class label equal to '1'. class 0 is the background class [25].

3.2.5. Data augmentation

In the transformation pipeline, we use several techniques to enhance the data set. Changes in lighting and orientation allow the model to learn to recognize objects under different conditions. The added blur provides an additional layer of security to the data. The image augmentations were executed using the Python library albumentations [26].

- **Horizontal Flip**

  This step flips the image horizontally - from left to right - with a certain probability. The default probability of 50% was chosen. An example image and image with applied transformation is shown in Figure 3.30.

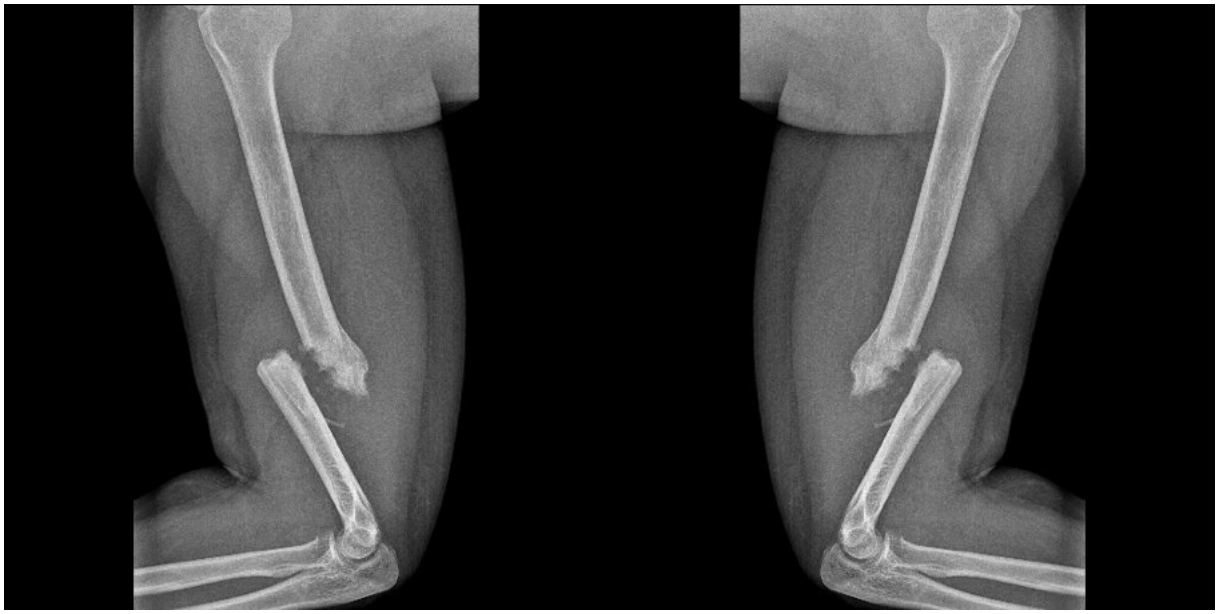

Figure 3.30: Left: original image, Right: image with horizontal flip transformation applied.

- **Vertical Flip**

  Similarly to horizontal flip, vertical flip flips the image vertically with certain probability. The default probability of 50% was chosen. An example image and image with applied transformation is shown in Figure 3.31.

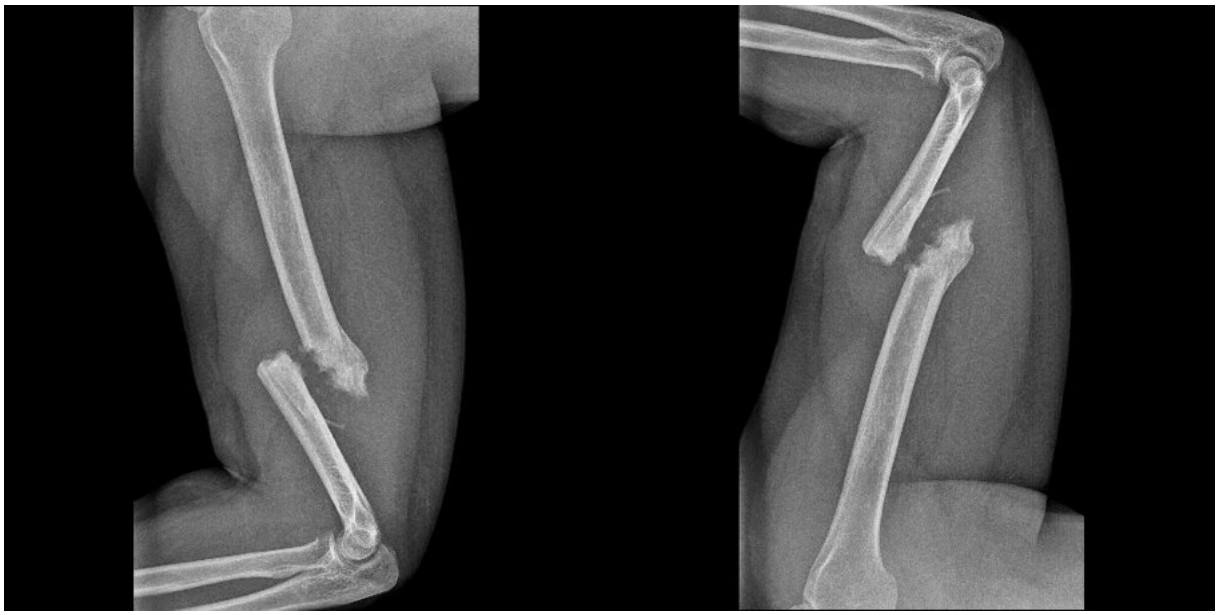

Figure 3.31: Left: original image, Right: image with vertical flip transformation applied.

- **Random Rotation**

  It randomly rotates the image by a random multiple of 90 degrees (90, 180 or 270 degrees). The default probability of rotation was 50%. An example image and image with applied transformation is shown in Figure 3.32

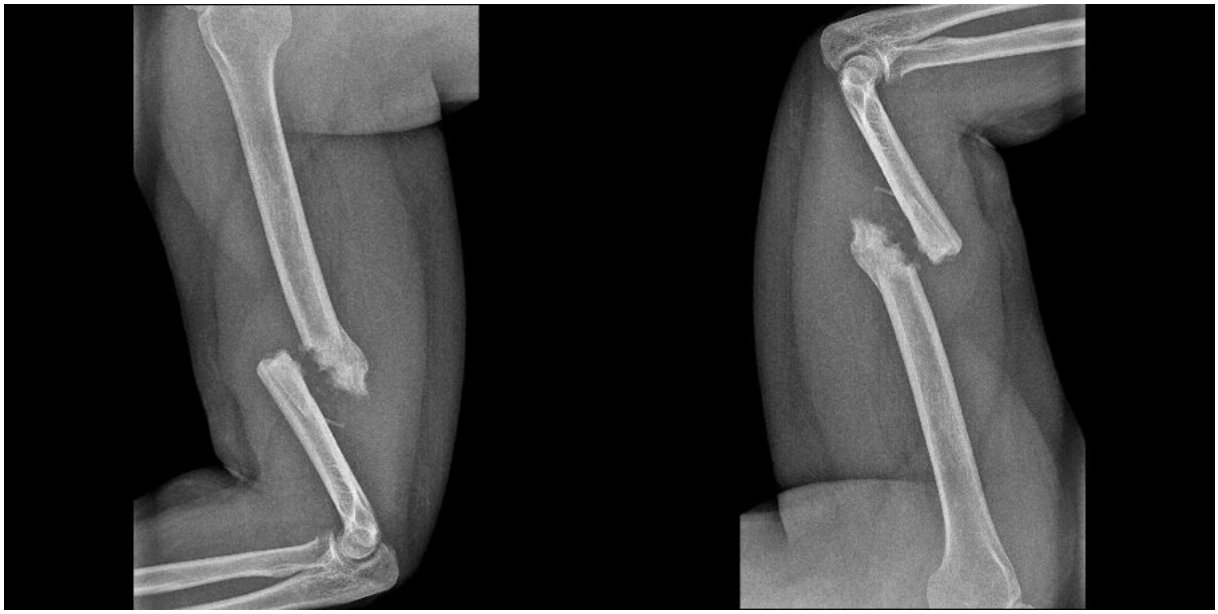

Figure 3.32: Left: original image, Right: image with random rotation transformation applied.

- **Random Brightness and Contrast**

  It randomly adjusts the contrast and brightness of the image with a default probability of 50%. The default values for brightness and contrast were applied. An example image and image with applied transformation is shown in Figure 3.33.

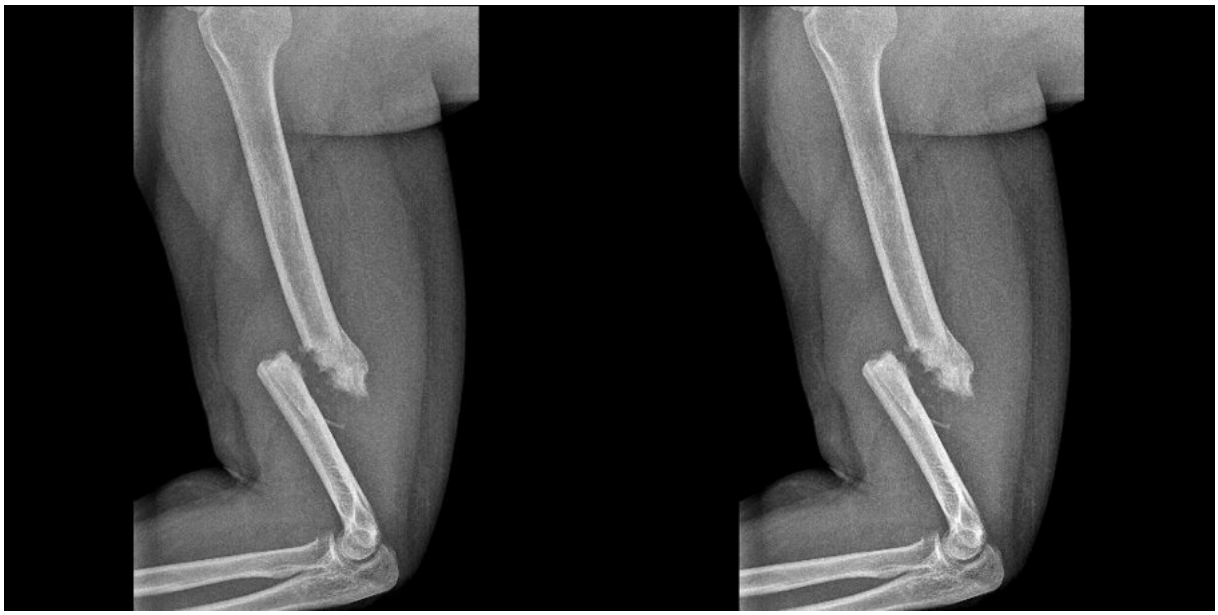

Figure 3.33: Left: original image, Right: image with random brightness and contrast transformation applied.

- **Gaussian Blur**

  This transformation adds additional Gaussian blur to the image with a default probability of 50%. An example image and image with applied transformation is shown in Figure 3.34.

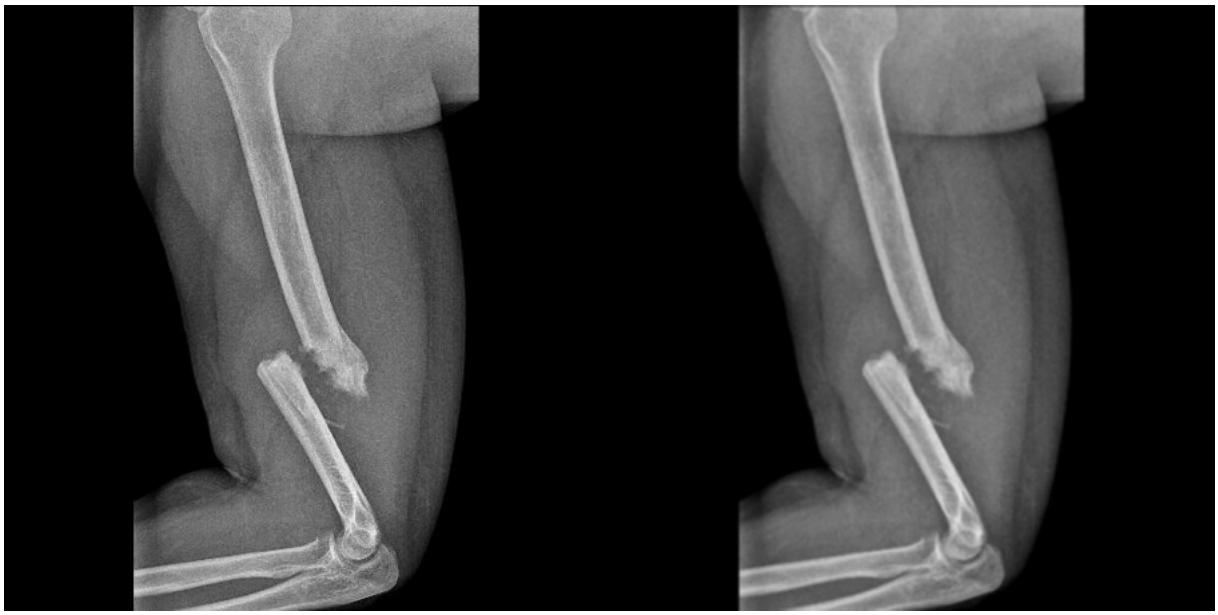

Figure 3.34: Left: original image, Right: image with gaussian blur added.

Each transformation is applied randomly with 50% probability. However, more than one transformation can be applied to a single image. The result of applying all transformations to a single image, which occurs only in a fraction of cases, is shown in Figure 3.35.

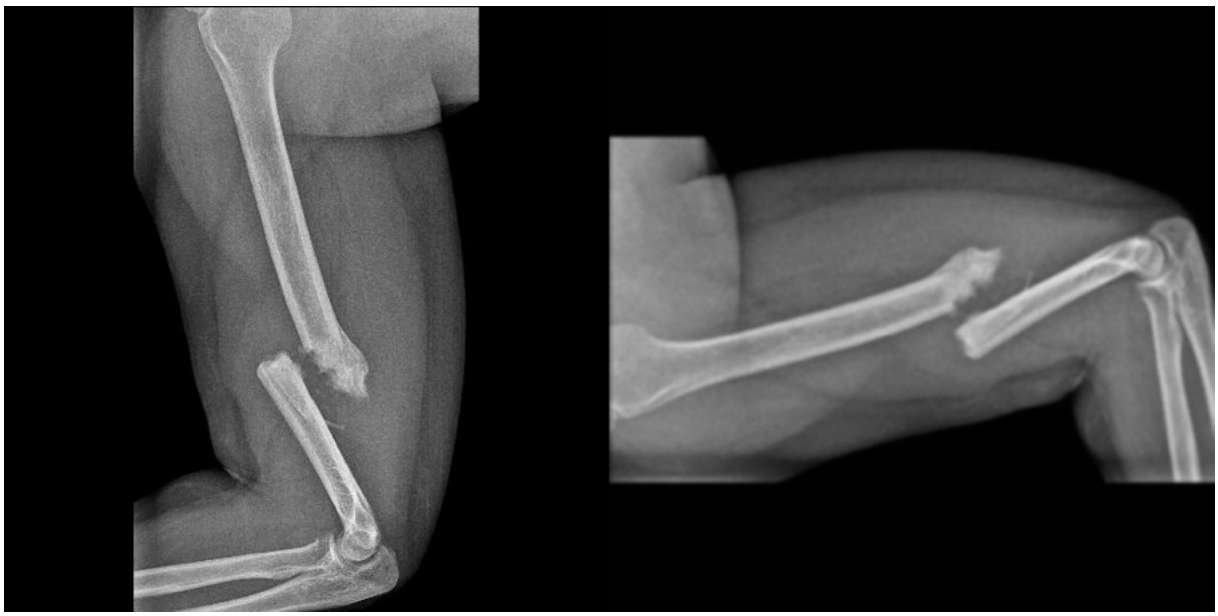

Figure 3.35: Left: original image, Right: image with all transformations applied.

3.2.6. Model architecture

a. Backbone with feature pyramid network

The backbone of the model is a ResNet-18 network enhanced with a Feature Pyramid Network (FPN) [16]. It allows one to build feature maps at multiple scales and enables the model to recognize objects of different sizes. Backbones creates feature maps which are later used to locate objects [14].

b. Region of Interest (RoI) align

The Region of Interest (RoI) Align component is used to accurately extract small feature maps from the multiple feature levels produced by the FPN. These extracted features correspond to the proposed object regions. RoI Align ensures that the features are correctly aligned with the input image, maintaining spatial coherence and improving the precision of the final detection boxes. This is particularly useful for handling objects at different scales and with various aspect ratios [14].

c. Region Proposal Network

The RPN takes the feature maps produced by the backbone and evaluates a set of predefined anchors (bounding boxes) to determine their likelihood of containing objects. These proposals are then passed on to the RoI Align layer for further refinement. The RPN is trained to efficiently generate high-quality object proposals, which significantly speeds up the detection process compared to traditional methods [14].

3.2.7. Data protection

Many techniques have been proposed to ensure that a malicious server owner or other training participants cannot calculate client data based on gradients sent to the server. Some techniques assume server can be trusted while others do not.

- **Secure Multi-Party Computation**

  SMPC provides cryptographic proof of security under certain conditions. It requires complex cryptographic operations that result in higher computational cost and higher communication bandwidth. Secure Multi-Party Computation allows clients to keep their data private while computing. However, this approach has high bandwidth requirements [27].

- **Secret Sharing**

  Secret sharing splits sensitive data into multiple shares which are then reconstructed when all invidual shares are merged. There are some challenges with this method, such as scalability issues and added complexity. If one or more participants are malicious, training or data privacy may be compromised [28].

- **Homomorphic encryption**

  Homomorphic encryption provides a way to perform computation on encrypted data. All participants share the same encryption and decryption keys, which is a substantial vulnerability. Multi-key homomorphic encryption provides more secure but more complex approach. Homomorphic encryption is another promising approach, but research is still ongoing [29].

- **Differential privacy**

  Differential privacy introduces noise at the local level into training data or gradients. It does not require trust in a centralized server. The amount of added noise is flexible and can be easily adjusted. Added noise may decrease model convergence speed and accuracy. Differential Privacy adds parameter controlled noise to the data which provides guarantees of its safety. Differential privacy does not require trust in a centralized server, but applies noise locally on the client side [27].

3.2.8. Differential privacy

In differential privacy within federated learning, noise from Gaussian distribution is most commonly used. Other noise types include Laplacian noise (where the noise is sampled from the Laplace distribution) and geometric noise for discrete data. More advanced noise types allow one to combine DP with encryption and secure aggregation. The noise is sampled from a Gaussian distribution with zero mean and standard deviation calibrated based on the desired privacy level. The amount of added Gaussian noise determines the level of privacy. Higher noise levels provide stronger privacy guarantees, as they make it harder for adversaries to infer any individual's data from the model updates. The parameter ε controls the privacy accuracy trade-off, with lower ε values indicating better privacy protection. Aggregation and iteration of the training process might mitigate some of the negative effects of noise. Higher noise levels affect the accuracy of the model and the convergence time. Achieving optimal privacy protection while maintaining acceptable model performance requires careful consideration to find balance between privacy and model accuracy. To achieve differential privacy, Gaussian noise is added to the model updates. The standard deviation σ of the Gaussian noise is calculated using the following formula:

$$\sigma = \frac{\sqrt{2\ln\left(\frac{1.25}{\delta}\right)}}{\varepsilon} \tag{3.4}$$

where:

- ε is the privacy budget parameter,
- δ is a small probability that allows for some relaxation in the privacy guarantee.

A Gaussian (or normal) distribution is characterized by its mean μ and standard deviation σ. The probability density function of a Gaussian distribution is given by:

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{3.5}$$

In the context of differential privacy, we use Gaussian noise with mean $\mu = 0$ and standard deviation σ calculated as above. This noise is added to model updates to maintain privacy [21].

The effect of the ε value on convergence is shown in Figure 3.36.

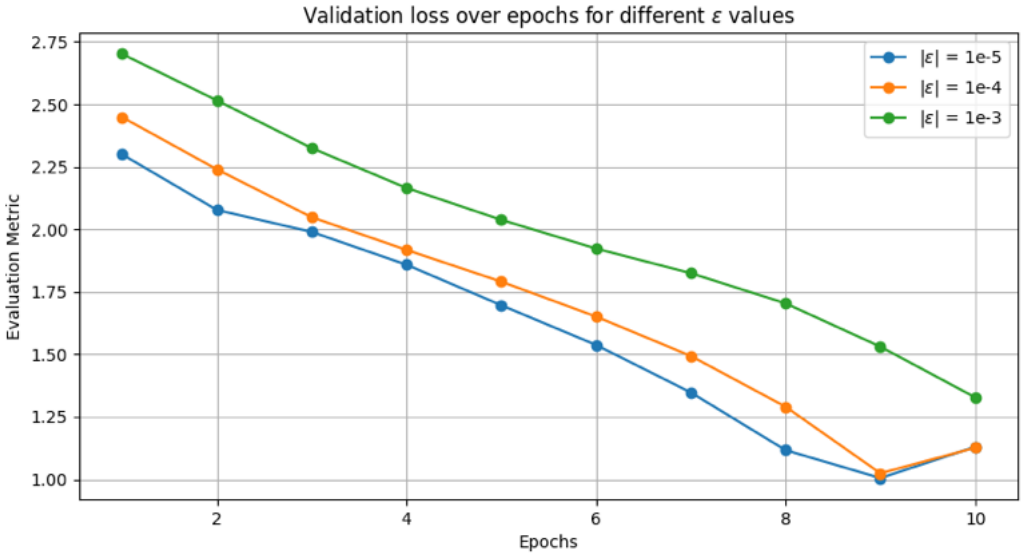

Figure 3.36: Training with different ε values. Higher ε results in more noise and slower convergence.

3.2.9. Training process

This section includes the results of the training process with respect to data heterogeneity. The learning data are not identically distributed, which presents challenges in model performance and generalization, necessitating specialized approaches to effectively handle and mitigate the effects of this heterogeneity.

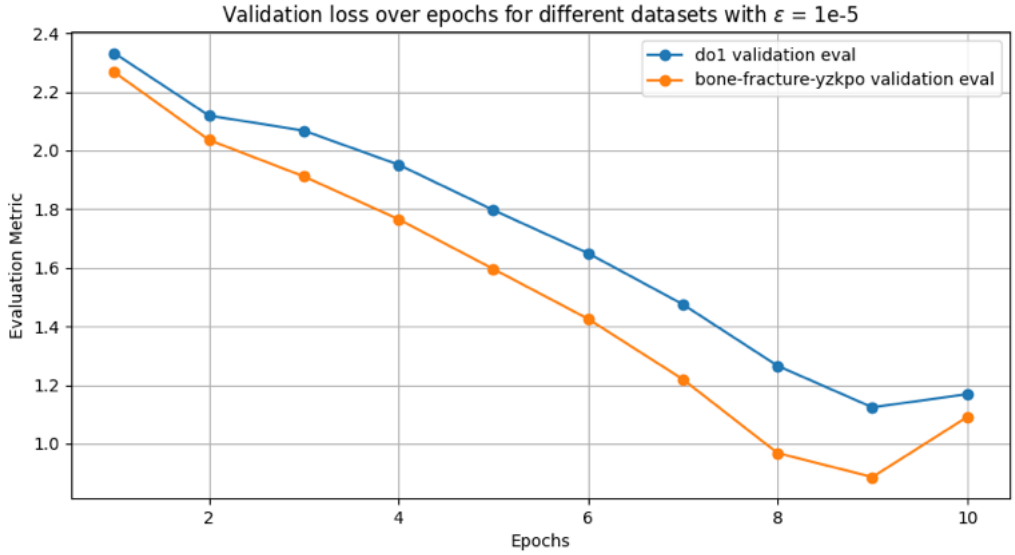

Figure 3.37: Validation loss over epochs with ε = 1e-5 for the two datasets do1 and bone-fracture-yzkpo.

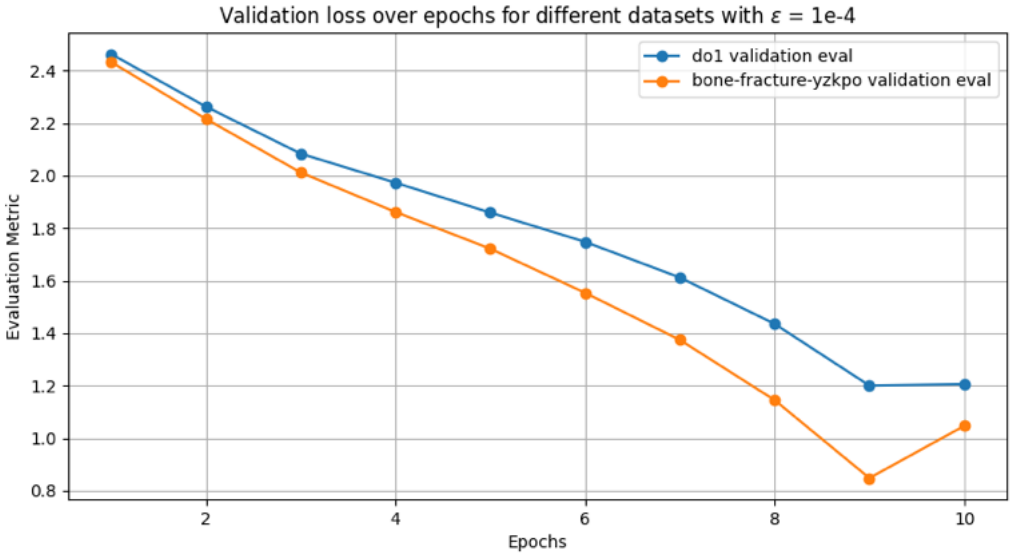

Figure 3.38: Validation loss over epochs with ε = 1e-4 for the two datasets do1 and bone-fracture-yzkpo.

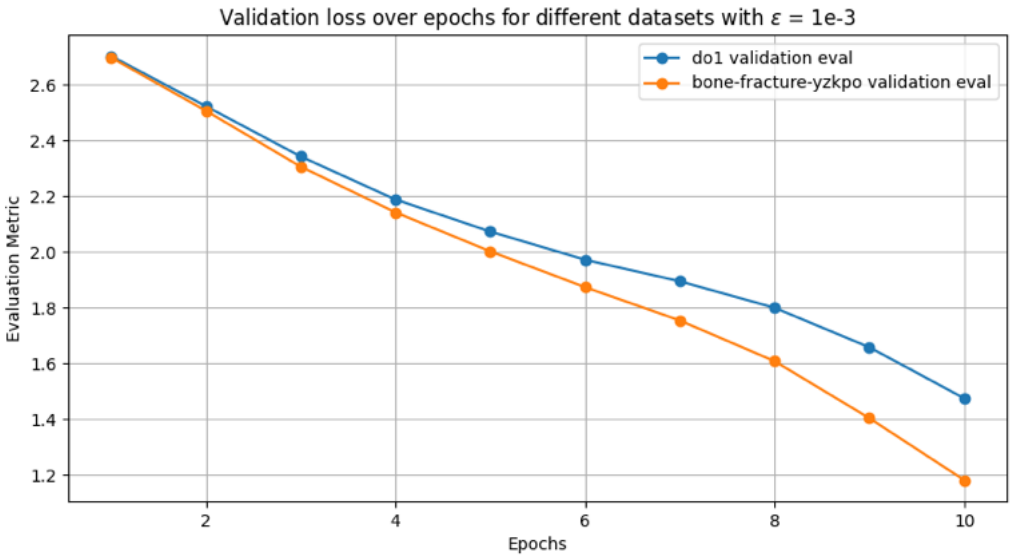

Figure 3.39: Validation loss over epochs with ε = 1e-3 for the two datasets do1 and bone-fracture-yzkpo.

For every chosen value of ε, the bone-fracture-yzkpo dataset (which has fewer images) consistently yields better results compared to the larger do1 dataset. While we expected the model to be biased towards the larger dataset, several factors can explain this outcome. One such factor is the better data quality in the smaller dataset. The bone-fracture-yzkpo dataset contains only images of wrist bone fractures, allowing the model to learn more effectively from these images. On the other hand, the high variability of do1 images might confuse the model, negatively impacting its performance.

3.2.10. Federated learning limitations and integration potential

Due to hardware limitations, the federated learning model was trained on a smaller dataset compared to the previous Faster R-CNN and YOLO models. As a result, this chapter focuses on showcasing the properties and potential of using federated learning with larger object detection models rather than achieving high detection accuracy with the current implementation.

Despite the lower accuracy of the federated learning model, it can still be integrated into the X-ray image interpretation system. This integration can be achieved by adding the model to the project's folder structure, which uses endpoints for sending images and receiving results. The system administrator can manage this integration, enabling future use of larger federated learning models with a higher accuracy.

3.3. Backend

3.3.1. Core technologies

The application is built using the following key technologies:

- **Spring Boot:** App framework, provides embedded Tomcat server, REST API specification.
- **H2 Database:** Stores data, can be easily swapped for other DB.
- **Hibernate:** An object-relational mapping, manages entity interaction.
- **HTTP:** the foundation of communication on the web, handles API requests and responses.
- **CORS:** Allows communication with applications running on different origins (frontend).

3.3.2. Database structure

The database structure used in the application is described below through a list of tables and attributes. It is also illustrated with a diagram in Figure 3.40.

a. User table

- **user_id:** Primary Key, unique identifier for each user.
- **email:** Email address of the user (unique).
- **hashed_password:** Password that has been encoded.
- **first_name:** First name of the user.
- **created_at:** Timestamp of creation.
- **updated_at:** Timestamp of the last update.
- **role:** patient or doctor. Defines the user type.

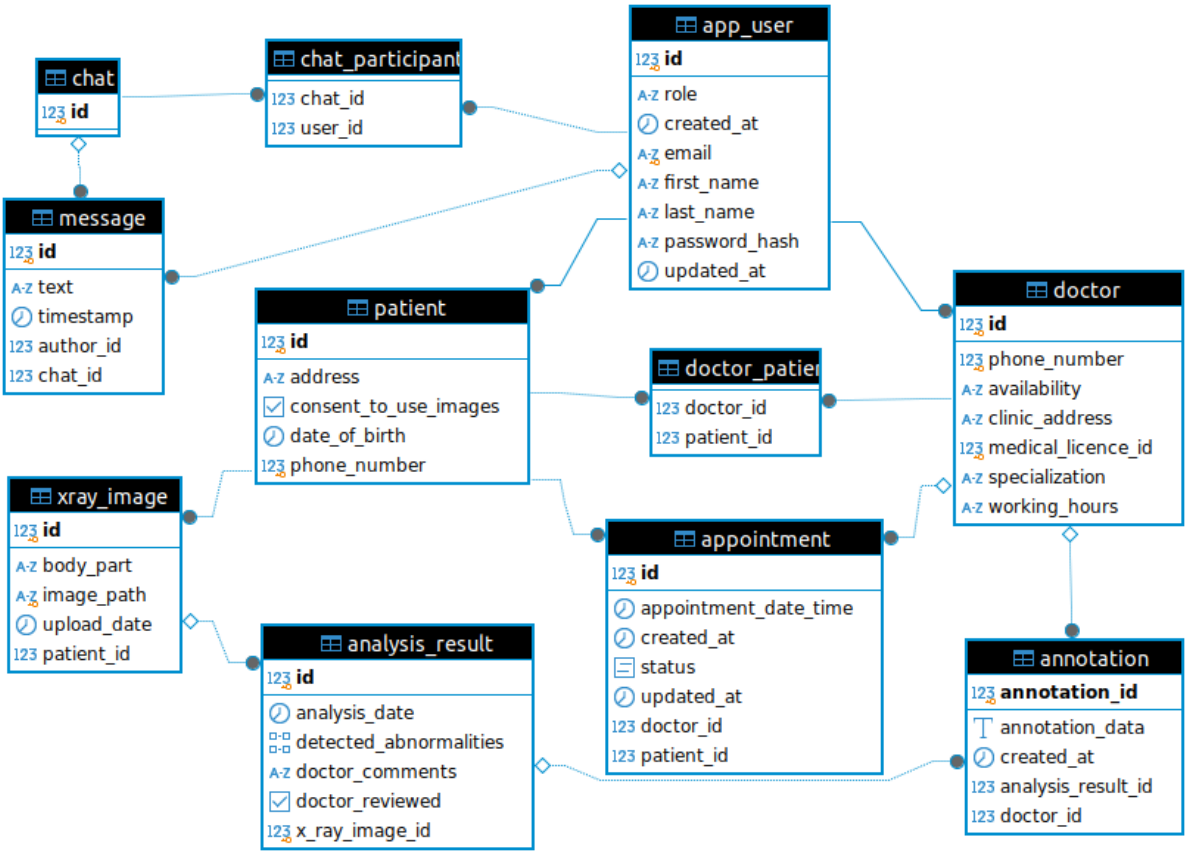

Figure 3.40: The backend application employs a relational database to manage and store data effectively. There are 11 tables in the database, their structure and relationships between the tables are detailed below.

b. Doctor table

- **user_id:** Primary Key and Foreign Key, referencing the **user_id** in the **User** table.
- **medical_licence_id:** Unique medical license number.
- **phone_number:** Contact number (unique).
- **clinic_address:** Address of the doctor's clinic.
- **specialization:** Area of medical expertise.
- **availability:** Days and times the doctor is available.
- **working_hours:** Typical working hours.

c. Patient table

- **user_id:** Primary Key and Foreign Key, referencing the **user_id** in the **User** table.
- **date_of_birth:** Date of birth of the patient.
- **address:** Residential address of the patient.
- **phone_number:** Contact number of the patient.
- **consent_to_use_images:** Boolean value indicating consent for image usage.

d. Doctor-Patient (join table)

- **doctor_id:** Foreign Key; references the ‘Doctor‘ table.
- **patient_id:** Foreign Key; references the ‘Patient‘ table.

The **Patient** and **Doctor** tables are linked to the **User** table through a One-to-One relationship, utilizing a shared primary key **user_id**. This approach follows the **Joined Inheritance Strategy**, where common user attributes such as email, password, first name, and timestamps are stored in the User table, while domain-specific data for patients and doctors is kept in their respective tables. This strategy provides a clean separation of shared attributes and domain-specific details, making the database schema more normalized and easier to extend.

An alternative approach is the **Single Table Inheritance Strategy**, where all data for users—regardless of type—is stored in a single table, User. This table would contain columns for both patient-specific and doctor-specific fields, leaving unused fields as NULL for records where they are not applicable. Additionally, there is need to define different constructors for user entities, as they will not use entire row of data, and set a column to discriminate between them.

The joined strategy, as used here, ensures a cleaner structure and better separation of concerns, at the cost of requiring additional JOIN operations to retrieve related data. Conversely, the Single Table strategy prioritizes simplicity and query performance at the expense of normalization. The choice between these two depends on the application’s complexity and performance requirements.

e. XRayImage table

- **image_id:** Primary Key; unique identifier for each image.
- **patient_id:** Foreign Key; references the ‘Patient‘ table.
- **body_part:** Body part represented in the X-ray image.
- **upload_date:** Timestamp of when the image was uploaded.
- **image_path:** File path of the stored image.

f. AnalysisResult table

- **analysis_id:** Primary Key; unique identifier for each analysis.
- **xray_image_id:** Foreign Key; references the ‘XRayImage‘ table. Analyzed image
- **detected_abnormalities:** List of abnormalities detected in the X-ray.
- **analysis_date:** Timestamp of when the analysis was performed.
- **doctor_reviewed:** Boolean value indicating whether the analysis has been reviewed by a doctor.
- **doctor_comments:** Comments from the reviewing doctor.

g. Annotation table

- **annotation_id:** Primary Key; unique identifier for each annotation.
- **analysis_id:** Foreign Key; references the ‘AnalysisResult‘ table.

- **doctor_id:** Foreign Key; references the ‘Doctor‘ table.
- **annotation_data:** JSON object with custom annotation data, stored as text.
- **created_at:** Timestamp of creation.

h. Appointment table

- **appointment_id:** Primary Key; unique identifier for each appointment.
- **patient_id:** Foreign Key; references the ‘Patient‘ table.
- **doctor_id:** Foreign Key; references the ‘Doctor‘ table.
- **appointment_datetime:** Date and time of the appointment.
- **status:** Enum indicating the status of the appointment (scheduled, canceled, completed).
- **created_at:** Timestamp of creation.
- **updated_at:** Timestamp of the last update.

i. Chat table

- **chat_id:** Primary Key, a unique identifier for each chat.
- **name:** The name or title of the chat (nullable for private chats).
- **created_at:** Timestamp indicating when the chat was created.

j. ChatParticipants table (join table)

- **chat_id:** Foreign Key, references the ‘Chat‘ table.
- **user_id:** Foreign Key, references the ‘User‘ table.

k. Message table

- **message_id:** Primary Key, a unique identifier for each message.
- **chat_id:** Foreign Key, references the ‘Chat‘ table.
- **author_id:** Foreign Key, references the ‘User‘ table.
- **text:** The content of the message, stored as a string.
- **timestamp:** Timestamp indicating when the message was sent.

l. Storing photos outside the database

In the system, the X-ray images are stored as files in a dedicated directory on the server rather than within the database. Only the file paths (URLs) are saved in the database. This approach is used for several reasons:

- **Efficiency:** Storing large binary files in a relational database can lead to performance degradation. Storing only the path ensures the database remains lightweight.
- **Scalability:** The file system is more scalable for handling large volumes of image data.

- **Ease of Access:** Files are stored in a structured directory `uploads/images/`, making them easily accessible for serving or further processing.
- **Flexibility:** Changes to the image storage system can be made without modifying the database schema.

It also seems functionally correct to store it as an external static resource. Data is stored in tables to be easily altered, and has to fit a certain pattern. Both are not true in relation to photo, which should never be changed due to internal backend operations, and generally be used in read-only mode. As for new photos, they can be vastly different from old (file format, size, resolution, contents) and still be valid, it would be an error to put them in one column.

When an X-ray image is uploaded, it is saved to the `uploads/images/` directory with a unique filename. The corresponding database entry in the **XRayImage** table stores metadata such as the patient ID, body part, upload date, and the file path.

This method optimizes system performance and ensures better manageability of multimedia data in the backend application.

3.3.3. Hibernate and JPA: Database integration with Java code

Hibernate is an Object-Relational Mapping (ORM) framework used in Java applications to interact with relational databases. It provides an abstraction layer over SQL, allowing developers to work with Java objects instead of writing complex SQL queries. Hibernate implements the Java Persistence API (JPA), which is a standard specification for ORM in Java.

a. Using Java code to create tables

Hibernate and JPA automatically generate database tables from Java classes annotated with JPA-specific annotations such as `@Entity`. These annotations define how a Java object maps to a database table, with fields in the class corresponding to columns in the table.

For example:

```
@Entity
@Table(name="app_user")
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name = "role", discriminatorType = DiscriminatorType.STRING)
public abstract class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Enumerated(EnumType.STRING)
    @Column(insertable = false, updatable = false)
    private Role role;
}
```

In the example above:

- The `@Entity` annotation marks the class as a JPA entity, which Hibernate maps to a table.
- The `@Table(name = "app_user")` annotation specifies the name of the table. Side note, name "user" is forbidden, as it is part of information schema.

- `@Inheritance(strategy = InheritanceType.JOINED)`: Specifies the inheritance mapping strategy. With `JOINED`, Hibernate creates separate tables for the base class (`User`) and each subclass, joined using foreign keys.
- `@DiscriminatorColumn(name = "role", discriminatorType = DiscriminatorType.STRING)`: Adds a `role` column in the `app_user` table to distinguish between subclasses.
- The `@Id` annotation indicates the primary key of the table.
- The `@GeneratedValue(strategy = GenerationType.IDENTITY)` annotation specifies that the primary key is auto-generated by the database.
- `@Enumerated(EnumType.STRING)`: Maps the `Role` enumeration to a string in the database.
- `@Column(insertable = false, updatable = false)`: annotation customizes the mapping of the field to a database column and prevents direct modification of the `role` field, as it is managed by the discriminator.

When the application starts, Hibernate reads the entity definitions and uses them to generate the corresponding database schema, including tables, columns, and relationships.

b. Database relationships

Hibernate supports advanced relational mappings such as `@OneToOne`, `@ManyToOne`, and `@ManyToMany`, which enable the creation of relationships between tables based on Java object relationships. For example:

```
@Entity
public class Doctor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @OneToMany(mappedBy = "doctor", cascade = CascadeType.ALL)
    private List<Appointment> appointments;


    @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE})
    @JoinTable(
            name = "doctor_patient", // Name of the join table
            joinColumns = @JoinColumn(name = "doctor_id"),
            inverseJoinColumns = @JoinColumn(name = "patient_id"),
        uniqueConstraints = @UniqueConstraint(columnNames = {"doctor_id", "patient_id"}
    )
    private List<Patient> patients;
}
```

Here:

- The `@ManyToOne` annotation defines a relationship where multiple appointments can be linked to a single doctor.

- Hibernate automatically generates foreign key constraints in the database based on these relationships.
- `@ManyToMany`: Defines the many-to-many relationship between the `Doctor` and `Patient` entities. The realtion table will be created automatically.
- `cascade = {CascadeType.PERSIST, CascadeType.MERGE}`: Specifies that changes to the `Doctor` entity (persist or merge operations) are cascaded to the associated `Patient` entities. This ensures that updates to the relationship are handled efficiently.
- `@JoinTable`: Specifies the details of the join table used to manage the many-to-many relationship.
  - **name**: The name of the join table, in this case, `doctor_patient`.
  - **joinColumns**: Defines the foreign key column in the join table that references the `Doctor` entity. The column is named `doctor_id`.
  - **inverseJoinColumns**: Defines the foreign key column in the join table that references the `Patient` entity. The column is named `patient_id`.
  - **uniqueConstraints**: Ensures that each combination of `doctor_id` and `patient_id` is unique, preventing duplicate entries in the join table.

c. Repositories write queries

In the context of Hibernate and JPA, repository classes provide the interface for database operations. They are implemented using the **Spring Data JPA** framework, which abstracts SQL queries and provides a clean, declarative way to interact with database tables.

By extending the `JpaRepository` interface, repositories gain access to common CRUD operations like `findById`, `save`, and `delete`, while also supporting custom queries using the `@Query` annotation or method name conventions. This reduces boilerplate code and leverages Hibernate's ability to translate Java methods into SQL.

Each repository corresponds to an entity and indirectly to a database table. When one entity is separated between 2 tables, as in the Patient-User case, JOIN operation is necessary. On repository level it is done by one repo extending the other, and so inheriting all its methods. The Patient Repository creates its Queries to the Patient table with inner joined User table, thus getting all data needed to create a Patient object.

Inherited repository declaration:

```
public interface UserRepository<T extends User> extends JpaRepository<T, Integer> {
}

public interface PatientRepository extends UserRepository<Patient> {
    List<Patient> findPatientsByDoctorId(int doctorId);
    ...
}
```

These repositories integrate seamlessly with the service layer to provide database access without exposing persistence logic to controllers.

The method findPatientsByDoctorId translates to the following SQL query:

```
SELECT p.*, u.*
FROM patient p
JOIN user u ON p.user_id = u.id
WHERE p.doctor_id = <doctorId>;
```

Then loops through the response and calls the Patient constructor with current row as arguments, thus creating a list of Patient objects.

d. SQL dialect universality

Hibernate is SQL dialect-independent, meaning it can work with a variety of relational database systems by using appropriate SQL dialects. For instance:

- The same Java code can be used with H2 (embedded database) for development and testing.
- In production, the H2 database can be replaced with PostgreSQL, MySQL, or another database system by simply changing the configuration properties. For example:

```
# Example H2 configuration (development)
spring.datasource.url=jdbc:h2:file:./data/new_db
spring.datasource.driver-class-name=org.h2.Driver

# Example PostgreSQL configuration (production)
spring.datasource.url=jdbc:postgresql://localhost:5432/mydatabase
spring.datasource.driver-class-name=org.postgresql.Driver
```

This flexibility ensures that the application remains portable across different environments without modifying the core Java code.

e. Benefits of using Hibernate and JPA

- **Abstraction**: Developers can focus on Java objects without worrying about database-specific SQL.
- **Portability**: The SQL dialect-independent design makes it easier to switch between databases.
- **Efficiency**: Reduces boilerplate code for managing database interactions.
- **Flexibility**: Supports complex mappings and database relationships.

3.3.4. API and endpoints

The backend application provides a RESTful API for client-server communication, adhering to standard practices to ensure scalability and maintainability. The API exposes endpoints for managing the system's core entities such as users, doctors, patients, annotations, and others. These endpoints are implemented using **Spring Boot** and follow the principles of REST (Representational State Transfer).

a. REST API design

The API follows the REST architectural pattern, utilizing HTTP methods to perform CRUD (Create, Read, Update, Delete) operations:

- **GET:** Used to retrieve resources, such as listing all patients or fetching a specific annotation.
- **POST:** Used to create new resources, such as registering a new doctor or adding a patient to a doctor.
- **PUT:** Used to update existing resources, such as modifying annotation data.
- **DELETE:** Used to remove resources, such as deleting a user or removing a patient from a doctor.

Each endpoint is uniquely identified by a URL, structured in a resource-oriented manner. For example:

- `GET /annotations` retrieves all annotations.
- `POST /annotations` creates a new annotation.
- `PUT /annotations/{id}` updates an existing annotation by ID.
- `DELETE /annotations/{id}` deletes an annotation by ID.

b. Endpoint implementation patterns

The API endpoints are implemented using Spring Boot's **@RestController** annotation, which automatically serializes Java objects into JSON format. Request mappings are defined using annotations such as `@GetMapping`, `@PostMapping`, `@PutMapping`, and `@DeleteMapping`. These annotations allow concise and intuitive endpoint definitions.

DTO (Data Transfer Object) patterns are used to decouple internal models from API responses. This ensures that only relevant and sanitized data is exposed to clients, enhancing security and flexibility. Mapping relations to DTO is essential, as it replaces whole related objects with its ids, thus reducing irrelevant data. If the object is needed it can always be accessed by its controller getById endpoint.

ResponseEntity is used to standardize HTTP responses, enabling the return of appropriate HTTP status codes and messages. For example:

- `ResponseEntity.ok()` for successful responses.
- `ResponseEntity.noContent()` for successful deletions.
- `ResponseEntity.badRequest()` for invalid requests.

c. OpenAPI and Swagger integration

The application integrates with **Springdoc OpenAPI**, which automatically generates API documentation based on endpoint annotations. This documentation is accessible via the Swagger UI, providing a user-friendly interface to test and explore the API. The endpoints are shown in Figure 3.41 as a screenshot from Swagger.

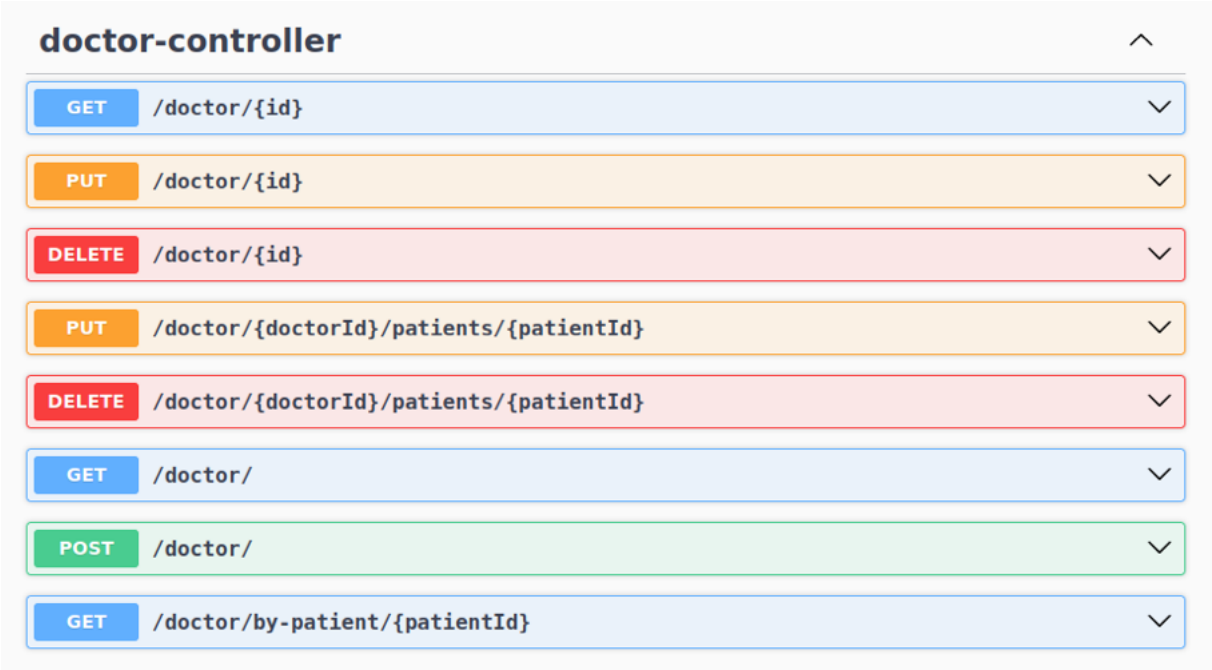

Figure 3.41: Visual representation of the doctor-controller endpoints generated by Swagger. Allows interaction with API's resources

d. Benefits of RESTful API design

The RESTful approach ensures that:

- The API remains stateless, enabling horizontal scalability.
- The uniform interface simplifies client-server interactions.
- Resource-oriented URLs and HTTP methods provide clarity and consistency.
- Clients are decoupled from server implementation, promoting flexibility in development.

By combining REST principles with OpenAPI-based documentation, the backend achieves a balance between functionality and ease of use for developers.

3.3.5. OpenAPI specification

The OpenAPI Specification is generated automatically using Swagger and SpringDoc. An example is shown in Listing 3.3.

```
openapi: 3.0.1
info:
  title: X-ray App API
  description: API documentation for the X-ray application.
  version: 1.0.0
servers:
  url:http://localhost:8080
  description:Generated server url
paths:
  /xray-images/{id}:
    get:
```

```yaml
      tags:
        - x-ray-image-controller
      operationId: getImageById
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
            format: int32
      responses:
        '200':
          description: OK
          content:
            '*/*':
              schema:
                $ref: '#/components/schemas/XRayImageDTO'
```

Listing 3.3: OpenAPI Specification snippet

OpenAPI provides a standardized and portable way to define and document REST APIs. Its primary advantage lies in its ability to serve as a contract between the backend and frontend systems. By describing endpoints, parameters, and responses in a machine-readable format, OpenAPI enables seamless integration and communication between development teams.

In particular, frontend developers can leverage the OpenAPI specification to understand and interact with backend endpoints without needing direct access to the backend code. Additionally, OpenAPI files can be used to auto-generate client-side SDKs, server stubs for testing. Any backend application, realising this specification can be substituted for this one, and still integrate with frontend safely.

The full OpenAPI specification for the API is available on GitHub and can be accessed as described in [30].

a. Data schema

Data schemas define the structure and format of the data exchanged between the client and the server in the application. In this context, schemas play a crucial role in ensuring that the data being sent or received adheres to a consistent structure, which is vital for both the integrity and validation of the application data.

The `XRayImageDTO` schema, as shown in Listing 3.4, defines the expected properties for an X-ray image object, such as `uploadDate` being a string. These field types ensure that the data matches the correct format and that any data that does not fit the specified schema will be rejected, preventing potential errors.

By using schemas, the backend can guarantee that the incoming data is structured properly before it is processed. Any discrepancies in data formatting will result in an error response, signaling that the client needs to adjust the data before resubmitting the request.

```
XRayImageDTO:
      properties:
        bodyPart:
          type: string
        id:
          format: int32
          type: integer
        patientId:
          format: int32
          type: integer
        uploadDate:
          format: date
          type: string
      type: object
```

Listing 3.4: OpenAPI Specification for XrayImageDTO schema

b. Image upload and download endpoints

The **GET /xray-images/{id}** endpoint retrieves an X-ray image file associated with a specific `id`. The response includes:

- **Content-Type**: The MIME type of the file (e.g., `image/jpeg` or `image/png`).
- **Content-Disposition**: Controls whether the file is displayed inline or downloaded.

The OpenAPI specification for this endpoint is shown in Listing 3.5.

```
{
  "get": {
    "tags": ["x-ray-image-controller"],
    "operationId": "getImageFile",
    "parameters": [{
      "name": "id",
      "in": "path",
      "required": true,
      "schema": {"type": "integer", "format": "int32"}
    }],
    "responses": {
      "200": {
        "description": "OK",
        "content": {"*/*": {"schema": {"type": "string", "format":
   "binary"}}}
      }
    }
  }
}
```

Listing 3.5: OpenAPI Specification for image file retrieval endpoint

The **POST /xray-images/{id}/upload** endpoint handles the upload of a single X-ray image file, associated with the image. Its metadata can be uploaded using **POST /xray-images/data**. The separation allows saving xrays that are yet to be taken, and later uploading the image, but there is also endpoint for doing both simultaneously.

The file is transmitted as a binary object in the request body. The `application/json` content type is used to wrap the file in a structured payload. This ensures compatibility with modern REST APIs and facilitates parsing by the backend. The schema enforces the following properties:

- **Type**: The file is defined as a `string` with the `binary` format.
- **Required Fields**: The `file` property is mandatory in the payload to ensure that incomplete requests are rejected.

The OpenAPI specification for this endpoint is provided in Listing 3.6.

```
{
  "post": {
    "operationId": "uploadImage",
    "parameters": [{
      "in": "path",
      "name": "id",
      "required": true,
      "schema": {"type": "integer", "format": "int32"}
    }],
    "requestBody": {
      "content": {
        "application/json": {
          "schema": {"type": "object", "properties": {"file": {"type":
    "string", "format": "binary"}}, "required": ["file"]}
        }
      }
    },
    "responses": {
      "200": {"description": "OK", "content": {"*/*": {"schema": {"type":
    "string"}}}}
    },
    "tags": ["x-ray-image-controller"]
  }
}
```

Listing 3.6: OpenAPI Specification for image upload endpoint

c. Reason for uploading single images

The system is designed to handle one image at a time, which is suitable for the workflow of medical imaging, where images are usually associated with individual patients. Benefits include:

- **Efficient Workflow**: Doctors or technicians can focus on specific images associated with individual cases or patients, reducing cognitive load.
- **Simpler Error Handling**: Errors during upload are isolated to individual files, making it easier to identify and resolve issues without disrupting other uploads.
- **Improved Frontend Rendering**: Displaying single images instead of a bulk collection ensures that the user interface remains responsive and avoids clutter.

3.4. Frontend architecture and implementation

3.4.1. Core technologies

The application is built using the following key technologies:

- **React.js:** A JavaScript library used for building reusable and dynamic user interface components, allowing for efficient rendering and management of application state.
- **React Router:** A tool for implementing client-side routing, enabling smooth navigation between pages without requiring a full page reload.
- **Lucide React:** A lightweight and flexible icon library used to provide consistent, visually appealing, and easily customizable icons across the application.
- **CSS Modules:** A styling approach that scopes styles locally to components, preventing conflicts and ensuring a clean and maintainable codebase.
- **Stream React Video SDK:** Used to enable real-time video consultations between patients and doctors, offering features like video and audio streaming, screen sharing, similar to those found in popular video conferencing applications.
- **Stream Chat React SDK:** Facilitates real-time messaging, allowing patients and doctors to exchange text messages, share multimedia, and maintain a seamless communication experience during and outside of video consultations.

3.4.2. Key components and their functionality

a. Login dashboard

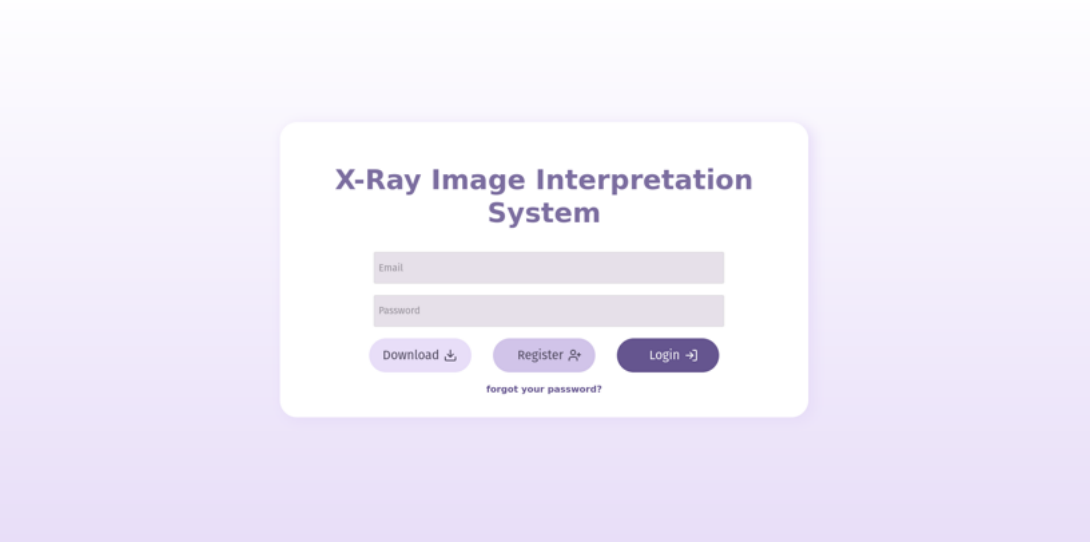

Figure 3.42: **Login Dashboard:** Provides a simple interface where users can enter their credentials to access the system. The form is validated to ensure both username and password are filled in before submission. The interface includes a download button that enables developers to access a curated dataset of consented X-ray images. These images have been specifically approved by patients for use in machine learning research, with all personal information removed.

b. Registration dashboard

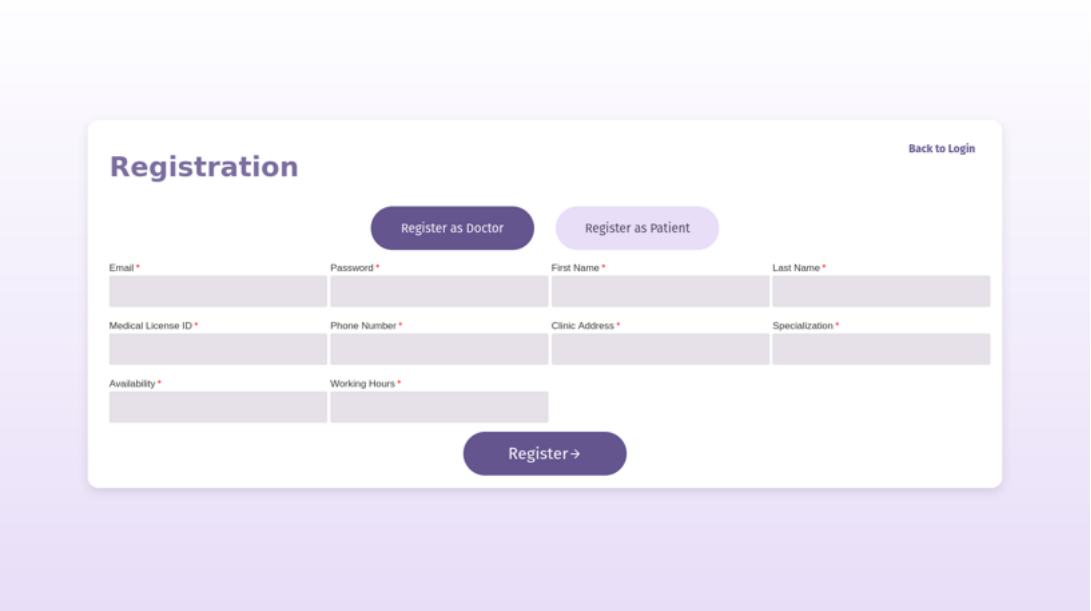

Figure 3.43: **Registration dashboard:** The Doctor registration window is tailored for healthcare professionals to register their profiles on the platform. It includes clearly labeled input fields and required inputs are marked with an asterisk. The interface supports validation, ensuring that users provide the necessary information before submission. Upon successful registration, doctors are redirected to the login screen.

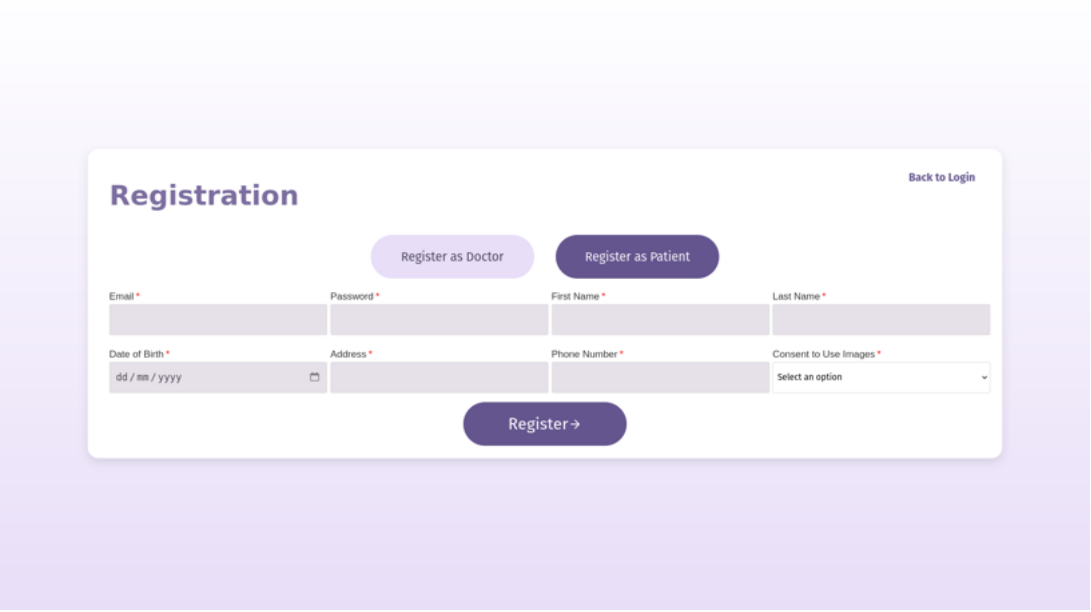

Figure 3.44: **Registration dashboard:** The Patient registration window is designed for patients to sign up and manage their profiles. It includes input fields similar to doctor's as well as an option to consent to the use of images for model training. Required fields are highlighted, providing clear guidance to users.

c. Patient dashboard

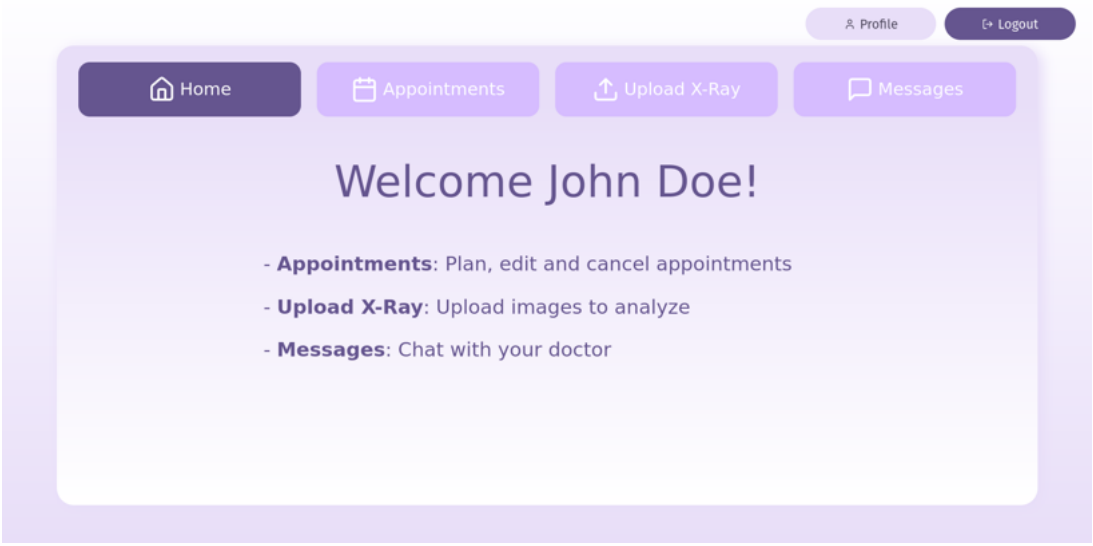

Figure 3.45: **Patient Dashboard:** The Home tab serves as the central landing page, welcoming users with a personalized greeting that displays their first and last name. This tab provides an intuitive overview of the system's core functionalities.

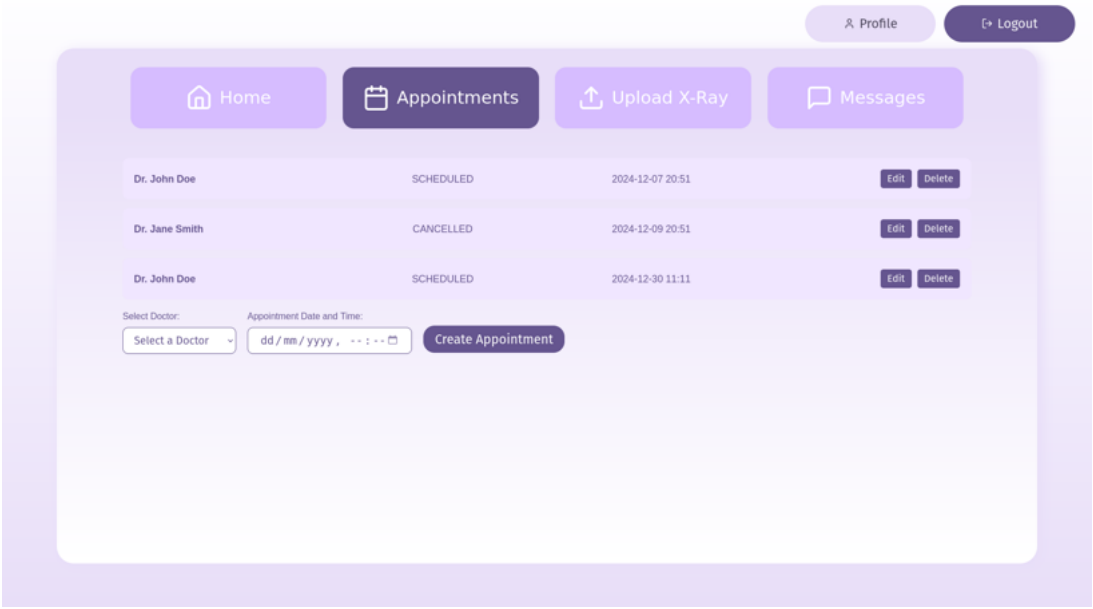

Figure 3.46: **Patient Dashboard:** The Appointments tab enables patients to maintain comprehensive control over their medical scheduling. It presents appointments in a structured grid layout, displaying essential information including doctor name, date, and time. Each appointment entry features interactive elements for editing and deletion, complemented by clear visual feedback through hover states.

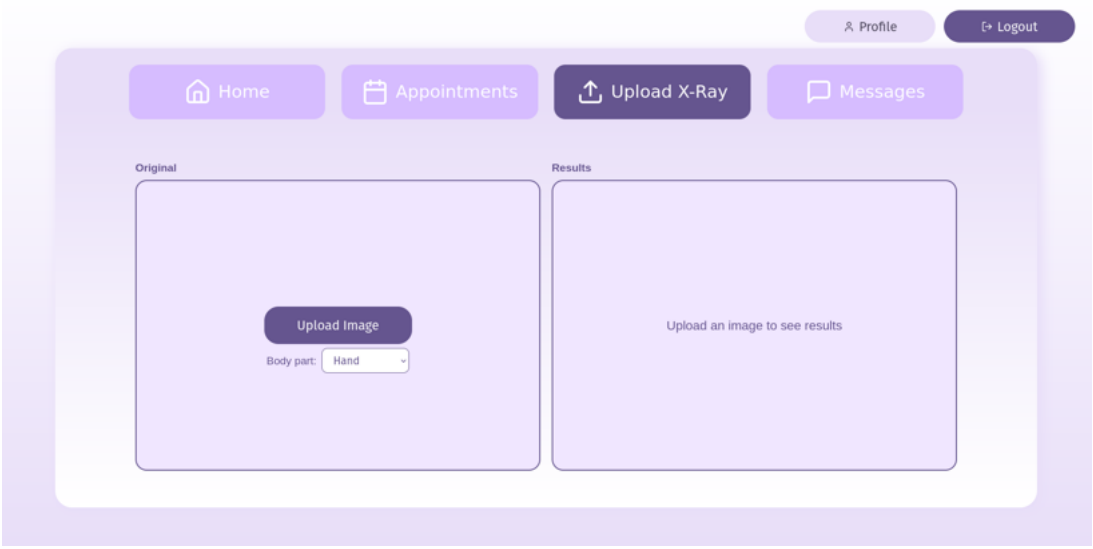

Figure 3.47: **Patient Dashboard:** The Upload X-ray tab is designed with a dual-panel layout that facilitates efficient image analysis workflow. The left panel serves as the upload area for original X-ray images, featuring a dropdown menu for precise body part selection and clear upload controls. The right panel is dedicated to displaying processing results, maintaining a clean separation of input and output.

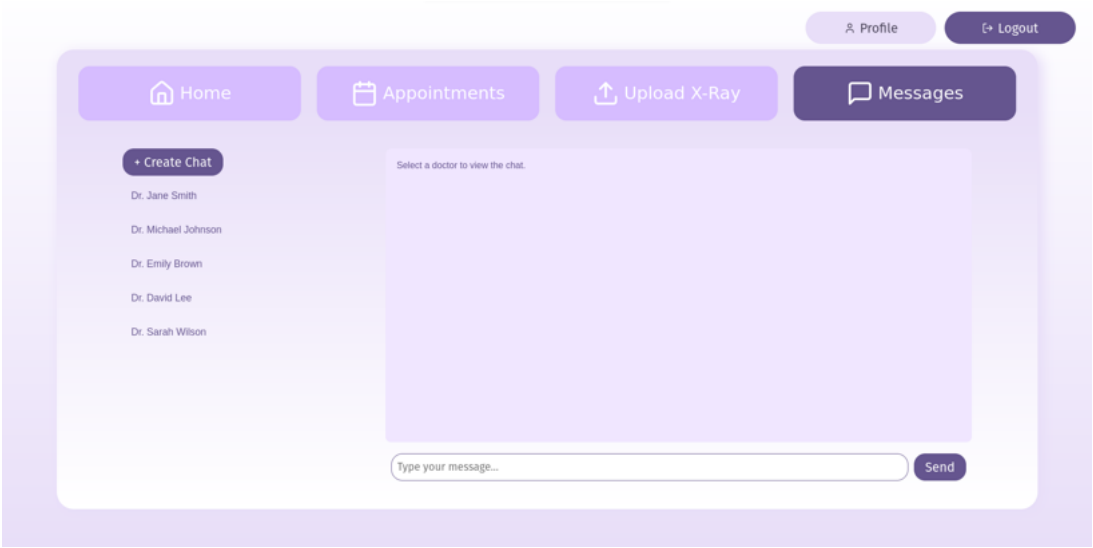

Figure 3.48: **Patient Dashboard:** The Messages tab implements a modern chat system layout with a split-view design. The left sidebar presents a scrollable list of healthcare providers, with active states clearly indicated through color contrast and bold typography. The main chat area features a message display region with clear sender identification and a fixed input area at the bottom. Additional functionality includes a prominent "Call" button in the chat header, enabling quick access to telehealth features.

d. Patient profile dashboard

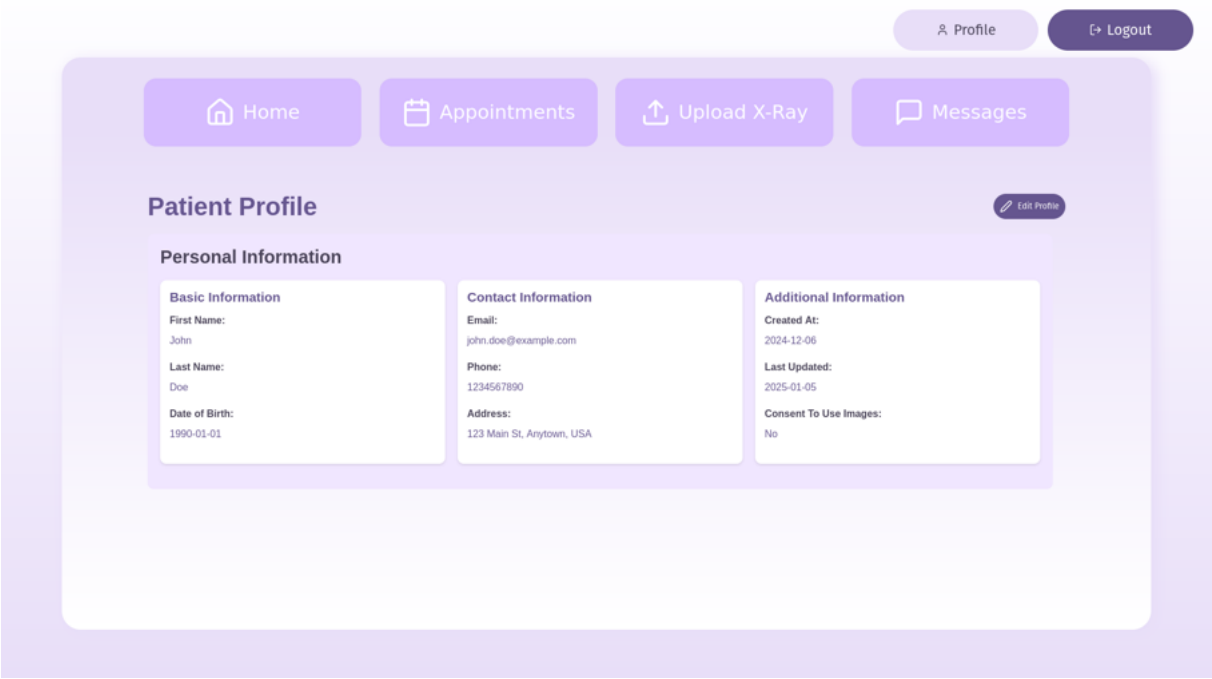

Figure 3.49: **Patient Profile Dashboard:** The Personal Information tab streamlines patient demographic and contact management through an intuitive interface. Users can easily update their basic information including name and birth date, manage contact details with validated email and phone fields, and maintain critical emergency contact information.

e. Doctor dashboard

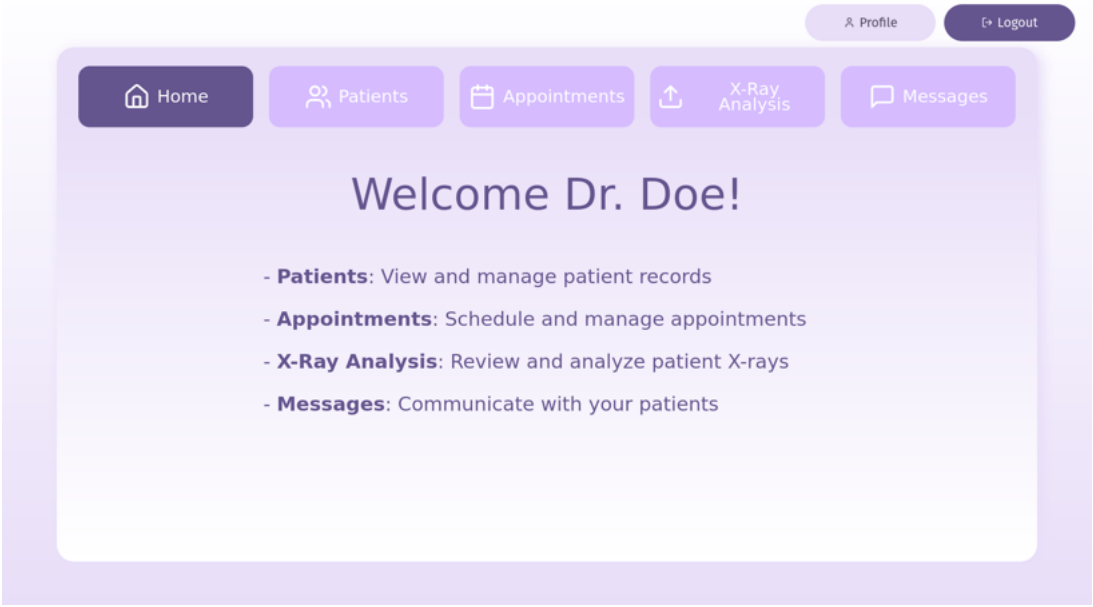

Figure 3.50: **Doctor Dashboard:** The Home tab serves as the landing area for doctors, providing an overview of available tools. This area centralizes critical information, ensuring doctors can quickly access their tasks and priorities.

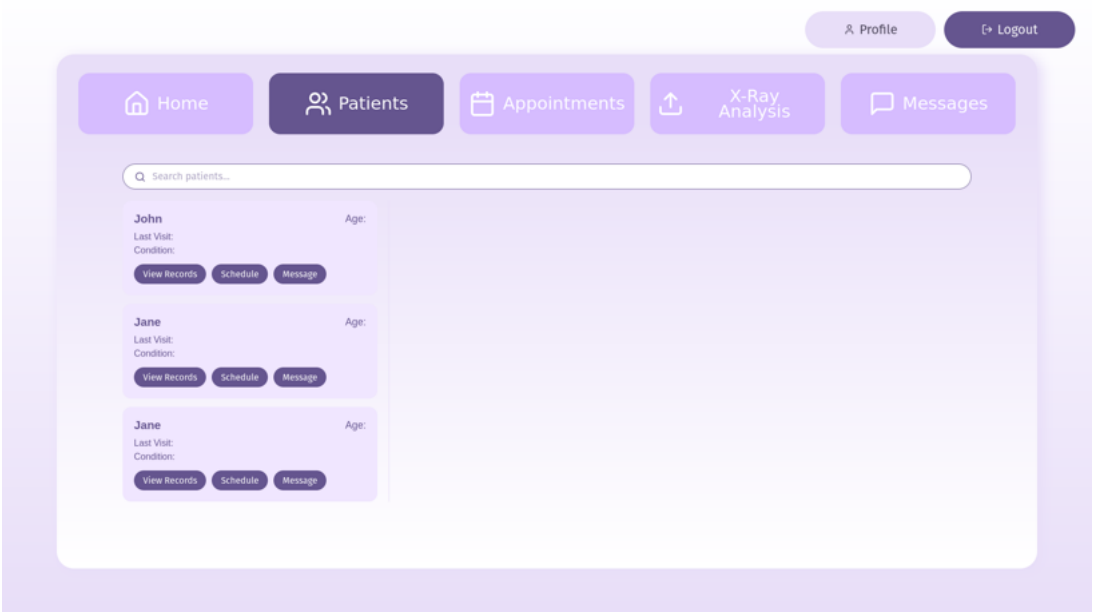

Figure 3.51: **Doctor Dashboard:** The Patients Tab enables doctors to view patients from a scrollable list. This feature includes search functionality and patient selection allowing quick filtering of patients by name as well as displaying detailed information and records for the selected patient.

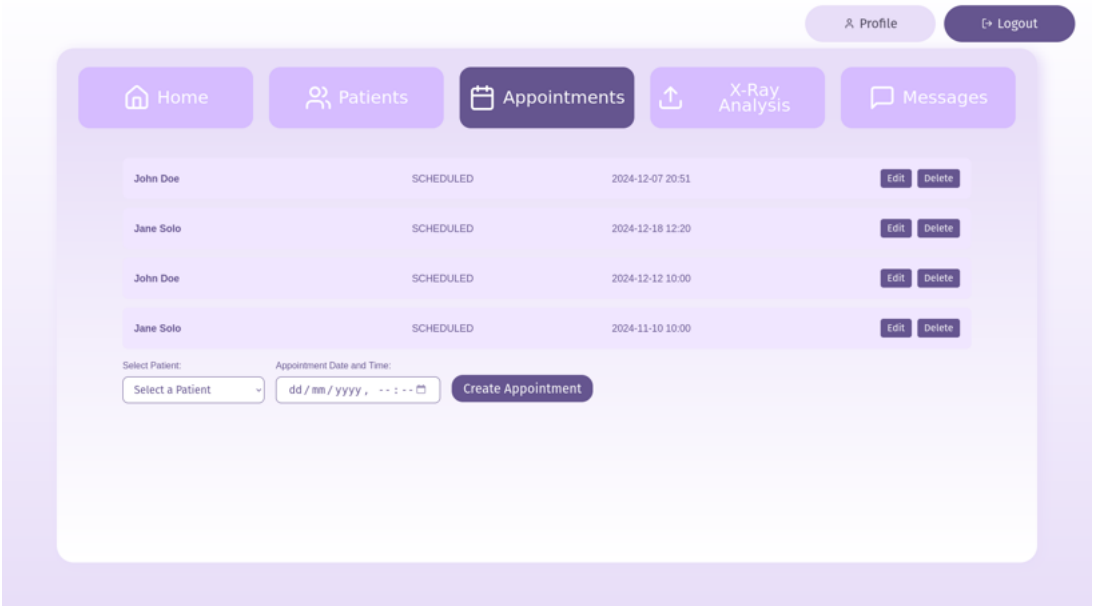

Figure 3.52: **Doctor Dashboard:** The Appointments tab provides an organized view of the doctor's schedule, showing all upcoming appointments with times, dates and patient names. Doctors can reschedule or cancel appointments directly from this interface.

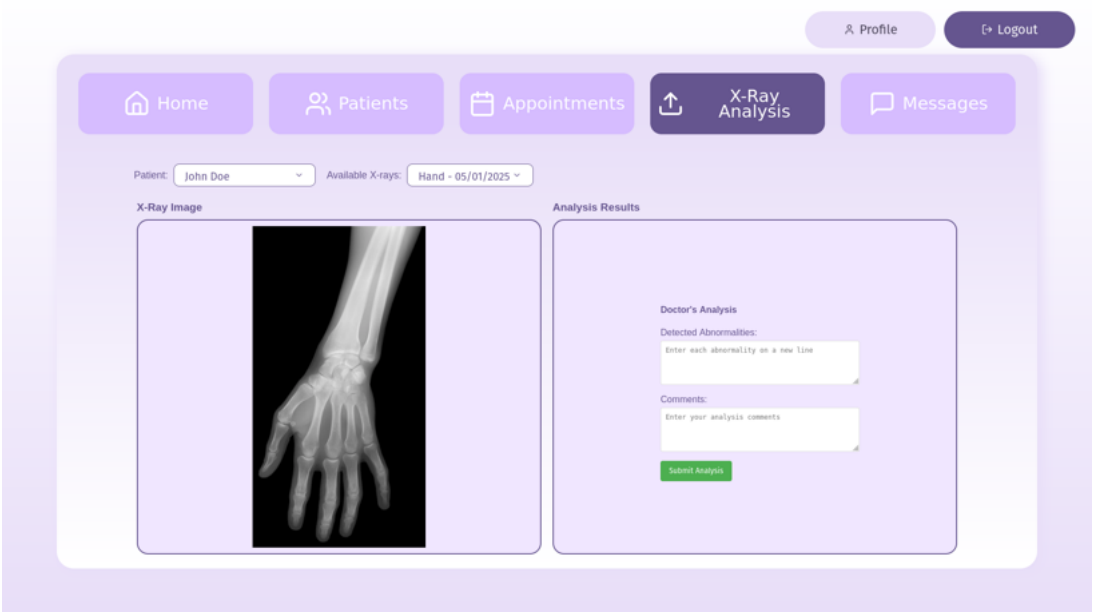

Figure 3.53: **Doctor Dashboard:** The X-ray Analysis Tab provides an interface for medical imaging analysis, empowering doctors with tools to view, analyze, and submit their findings. Left panel displays X-ray images uploaded by the selected patient. Once an X-ray image is selected, doctors can add detailed comments and submit their analysis directly on the right panel.

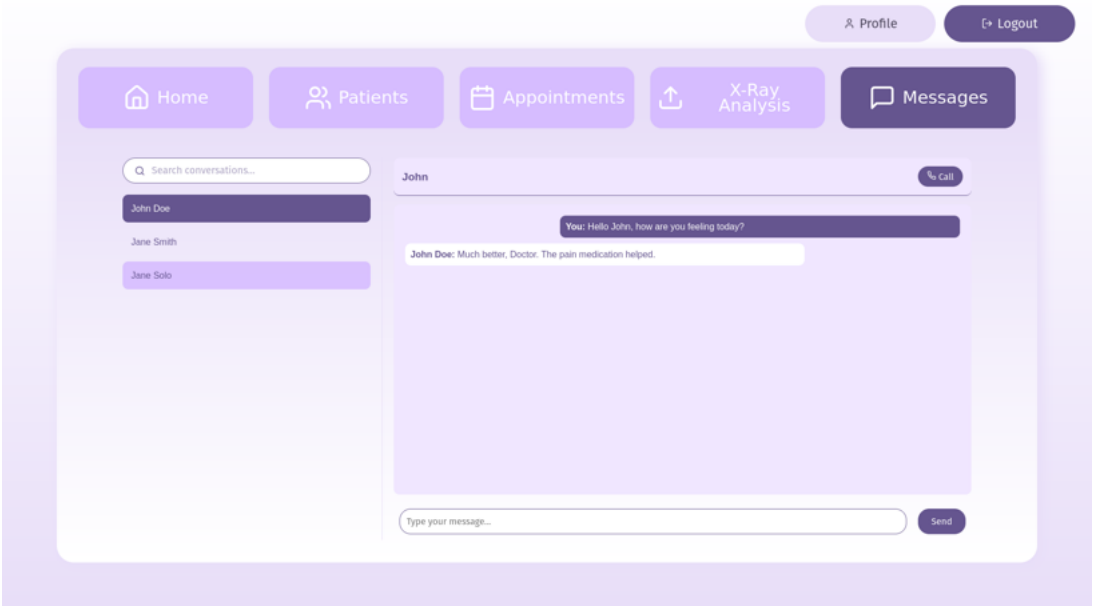

Figure 3.54: **Doctor Dashboard:** The Messages Tab in the Doctor Dashboard implements a modern communication system with a split-view design, tailored for efficient doctor-patient interaction. Left sidebar is a scrollable list of patients with whom the doctor is currently communicating. The central chat window features a message display region with clear sender identification, and distinct styling for doctor and patient messages. Positioned in the chat header, the "Call" button allows doctors to seamlessly initiate telehealth consultations.

f. Doctor profile dashboard

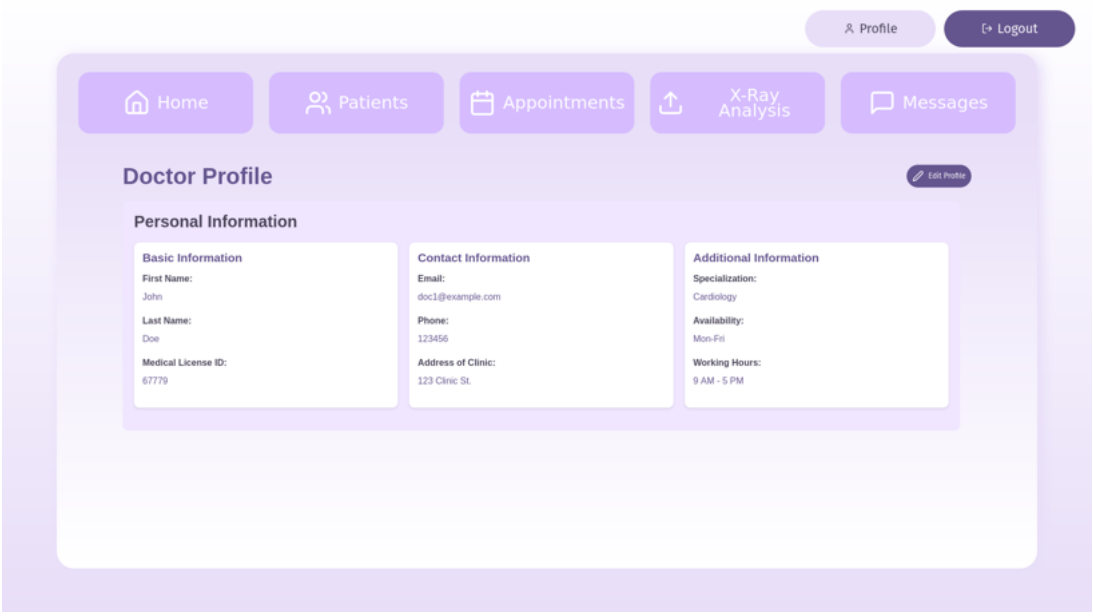

Figure 3.55: **Doctor Profile Dashboard:** The Personal Information tab offers a comprehensive and user-friendly interface for managing doctor profiles. It enables users to seamlessly update personal information and contact details. The dashboard also provides functionality to edit additional details, including availability, working hours, and clinic-related information. It simplifies profile management with a streamlined editing mode and real-time data synchronization.

g. Consultation room

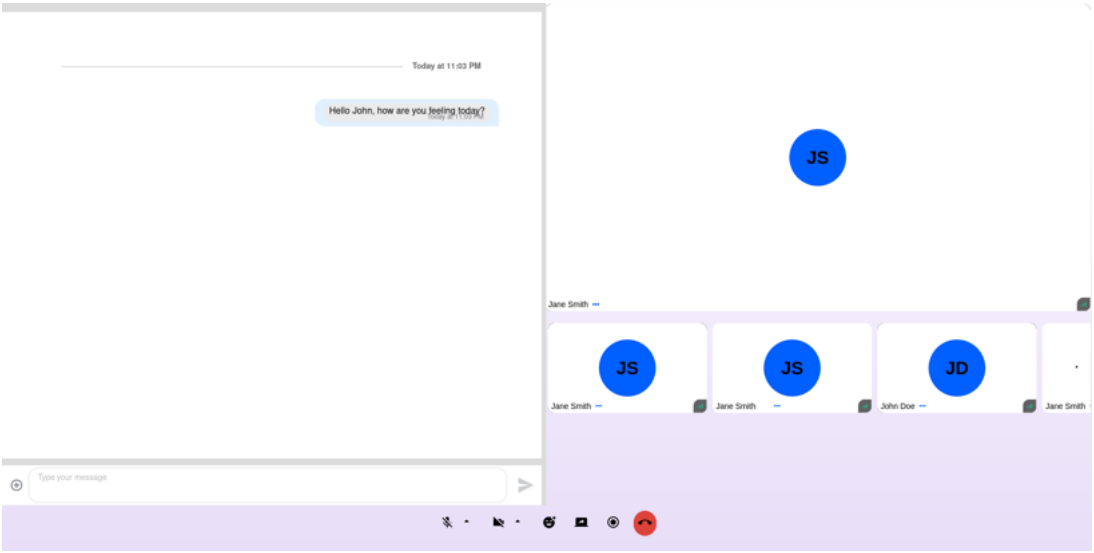

Figure 3.56: **Consultation Room:** This interface provides a fully integrated platform for real-time communication between patients and doctors. It combines a seamless chat interface, powered by the Stream Chat React SDK, with high-quality video call functionality enabled by the Stream React Video SDK. The split-screen layout allows simultaneous access to chat messages and live video feeds. The room also includes intuitive controls for managing calls, such as enabling/disabling cameras and microphones or leaving the call.

3.4.3. User interface design

The interface design emphasizes:

- **Visual Hierarchy:** Use of the consistent color scheme and Roboto font family for clear typography as well as intuitive navigation through a tab-based interface.
- **Responsive Design:** Utilization of viewport-relative units (vw, vh), flexible grid systems, adaptive component sizing and dynamic content organization.
- **Interactive Elements:** Enhanced user interaction through hover effects on interactive elements, clear visual feedback for active states, smooth transitions and animations, consistent button styling and behavior.

3.4.4. State management

The application leverages React's built-in state management capabilities, specifically the useState and useEffect hooks, to ensure a dynamic and responsive user experience.

The useState hook is used to manage the authentication state at the top-level App component. The isLoggedIn state determines whether the user is authenticated, while the userRole state tracks the role of the authenticated user (e.g., patient or doctor). These states are updated through the handleLogin function upon successful login. By maintaining these states in the App component, changes to the authentication status propagate throughout the application, dynamically affecting routing and access control.

The useEffect hook complements the useState hook by managing side effects and synchronizing application state with external resources. In the context of authentication, useEffect ensures that user sessions persist across page reloads. In role-specific dashboards, useEffect is also used to fetch data dynamically (e.g., appointments or messages) and update the state. This hook is also used to set up third-party services like the Stream Chat client, establishing connections when the component mounts and disconnecting them when the component unmounts.

3.4.5. Routing and navigation in the application

Routing is managed using React Router. The Routes component is used to define navigation paths. The primary paths include /login for the login page, /patient-dashboard and /doctor-dashboard for authenticated users based on their roles as well as /consultation/:consultationId with dynamic paths for managing different consultation rooms. The implementation of these paths is detailed in Listing 3.7.

In addition to declarative routing through Routes, the application uses React Router's useNavigate hook for programmatic navigation. The useNavigate hook enables dynamic redirection based on user interactions or application logic. The useNavigate hook is used in the handleRegister function to redirect users to the /register route when they opt to create a new account, as shown in Listing 3.8.

The useNavigate hook also powers the redirection to specific consultation rooms dynamically. In the handleCall function, after ensuring the necessary setup for a video call (e.g., generating a consultation room ID), the application navigates the user to the appropriate /consultation/:consultationId route. This process is illustrated in Listing 3.9.

```
<Router>
  <Routes>
    <Route path="/login"
        element={<LoginDashboard onLogin={handleLogin} />} />
    <Route path="/register" element={<RegistrationDashboard />} />
    <Route path="/patient-dashboard"
        element={<PatientDashboard onLogout={handleLogout}
        patientInfo={userInfo}/>} />
    <Route path="/doctor-dashboard"
        element={<DoctorDashboard onLogout={handleLogout}
        doctorInfo={userInfo} />} />
    <Route path="/consultation/:consultationId"
        element={<ConsultationRoom onLogout={handleLogout}
        userInfo={userInfo} userRole={userRole}/>} />
    <Route path="*"
        element={<LoginDashboard onLogin={handleLogin} />} />
  </Routes>
</Router>
```

Listing 3.7: Defining paths using React Router

```
const handleRegister = () => {
    navigate('/register');
};
```

Listing 3.8: Navigation to Registration dashboard

```
const handleCall = async () => {
    ...
    navigate(/consultation/consultationRoomId‘);;
```

Listing 3.9: Navigation to Consultation room

Chapter 4

Work organization

This project was carried out by a team of four and required thorough preparation as well as a clear division of tasks among the team members. Below is a brief description of each team member's role, the overall workflow of the project, and the tools used for planning and completing tasks.

4.1. Groups involved in the project

During the work on the thesis, it was possible to distinguish specific groups of people involved in its development:

- **Authors of the thesis:** Krzysztof Ćwiertnia, Kacper Kozik, Rafał Piwowar, Michał Sośnik
- **Thesis Supervisor:** Ph.D. Eng. Marcin Kuta
- **Coordinator of the Project Laboratory course:** Ph.D. D.Sc. Eng. Rafał Dreżewski

4.2. Project workflow

Before starting the work, everything was carefully planned. It was assumed that the tasks would be divided equally among the team members. Key milestones were established at the beginning, based on the checkpoint deadlines set in the Project Laboratory course. A hybrid approach in software development methodology was applied for the project, combining Agile principles with regular meetings involving team members and the client (Thesis Supervisor). At the beginning, the team decided on a gradual assignment of tasks among members. The application was developed based on feedback from the client, following an iterative and incremental development process characteristic of Agile methodology. Additionally, status reports were provided to the client after each stage of adding new functionality to the application or detection models for bone fractures.

Initially, the tasks involved defining the roles of team members. Each member then presented his ideas for the work they would be responsible for. After collectively determining the responsibilities, the team planned the first application prototypes and defined initial use cases that could potentially be implemented. Various functionalities were identified and divided among frontend development, backend development, and deep learning detection models. Then a risk analysis was created to address the potential limitations that might arise during the development of the application and the thesis.

One of the most important planning tasks was identifying the target groups for the project. All team members created user profiles and defined user stories based on these profiles. This was highly useful for creating prototypes of the application and distributing tasks. It also helped establish the scope of work and the structure of this technical documentation. Additionally, the non-functional requirements provided insights into the qualities a well-implemented system should have, reflecting the features expected of modern applications.

During this time, a task and workflow dashboard was created using Trello, an online project management tool. Backlogs were added, and tasks were assigned to the thesis authors. A snapshot of the Trello board with planned tasks is shown in Figure 4.1.

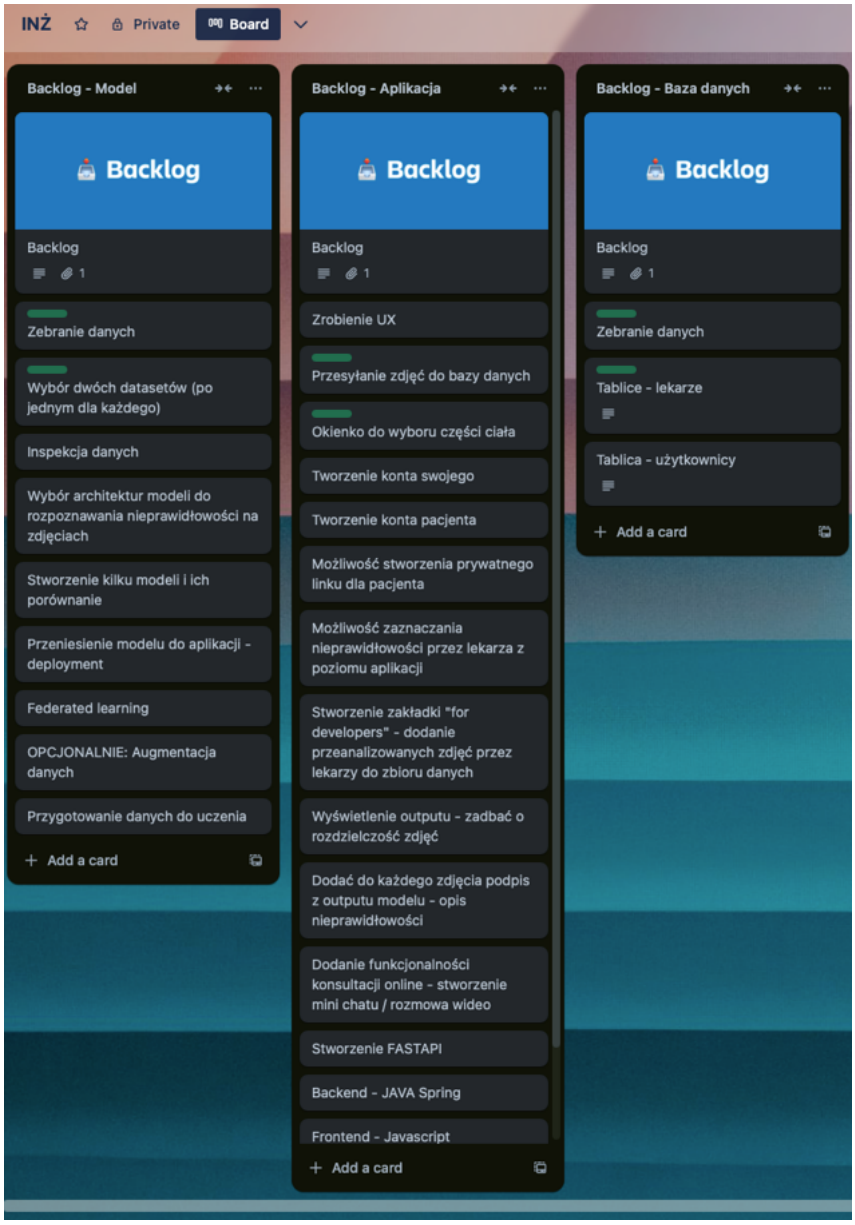

Figure 4.1: Snapshot of the Trello dashboard showing a part of planned tasks and workflows for the thesis project.

The entire task division was consulted with the Thesis Supervisor and the Coordinator of the Project Laboratory course. Both provided valuable feedback and confirmed that this was a good approach.

Next, Kacper Kozik and Rafał Piwowar, who were responsible for organizing datasets and implementing deep learning detection models, began searching for suitable data sources to train the models. Due to the wide range of potential human body parts that could be analyzed for bone abnormalities on X-ray images, the team decided to focus on the wrist and arms. This decision was based on two publicly available datasets. With a larger team or a company setting, it might have been possible to analyze more body parts. However, the goal of this thesis was to demonstrate that such tasks can be performed using modern machine learning models and could be expanded upon in the future.

Rafał Piwowar then focused on federated learning, including the application of differential privacy to protect data. He conducted experiments with different privacy levels and analyzed their impact on model accuracy and convergence time. Meanwhile, Kacper Kozik conducted an in-depth analysis of how various machine learning model hyperparameters influence detection accuracy, such as mAP, PR curves, precision, and recall.

For the development of the web application, Michał Sośnik and Krzysztof Ćwiertnia were the main contributors. Michał Sośnik was responsible for implementing the backend, including managing medical data, designing and implementing the database, and creating APIs to handle the transfer of X-ray images for automatic analysis and user interactions. After reviewing the initial database schema proposed by Kacper Kozik, Michał implemented the backend system. On the other hand, Krzysztof Ćwiertnia focused on the frontend development. After consulting with Michał Sośnik, Rafał Piwowar, and Kacper Kozik about the application's design and functionality, Krzysztof worked on implementing the user interface, including a chat and video call feature for communication between patients and doctors.

This division of work was guided by a hybrid methodology proposed by the team members. The approach involved separating the three main development areas: frontend, backend, and deep learning models for as long as possible. This allowed each team member to work independently on his respective tasks, ensuring individual contributions and ideas. This approach eliminated the need for strict deadlines before integrating the components. The team understood that allowing members to work independently for an extended period would lead to improved components, system, and overall work quality. However, even though tasks were separated, the team regularly did code reviews and implementation checks. This helped suggest new features based on team feedback on the product. The final integration of the system was systematically monitored to ensure alignment with the original objectives. Krzysztof Ćwiertnia was responsible for connecting the backend components with the frontend and finalizing the integration.

Apart from Trello, the team used a GitHub repository to gradually implement backend and frontend components [31]. Additionally, a separate GitHub repository was created to include the analysis of models used in this thesis [32]. These repositories contain the source code necessary for the system's development. Additionally, Discord, set up by Rafał Piwowar, served as the primary communication tool among the thesis authors. Figure 4.2 shows an example screenshot from the Discord conversation related to this thesis. It allowed communication, idea sharing, and task planning, which was primarily coordinated by Kacper Kozik.

For communication with the Thesis Supervisor, a Microsoft Teams channel was used (a screenshot of the Microsoft Teams channel is presented in Figure 4.3). Meetings and consultations were organized after achieving project milestones or when questions arose about the application's further development. These meetings were scheduled after checking the availability of all team members and the Supervisor. The authors would like to express their gratitude to Ph.D. Eng. Marcin Kuta for his valuable advices.

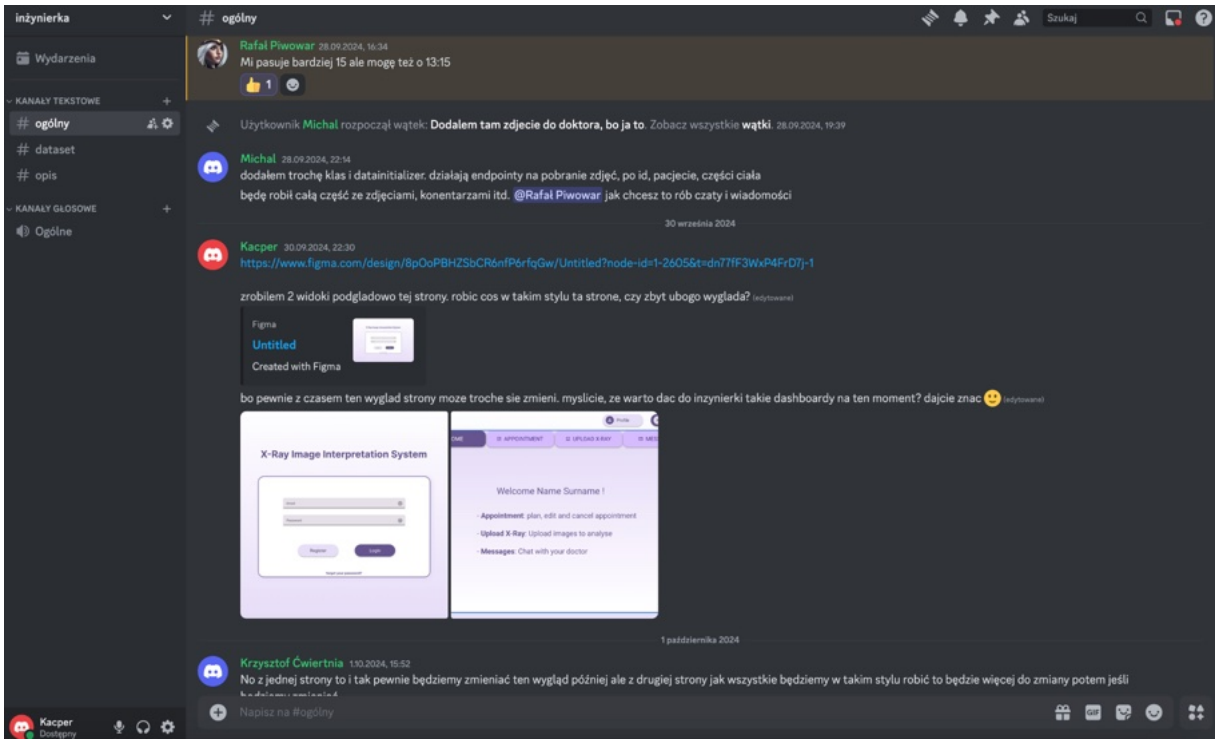

Figure 4.2: Snapshot of a Discord conversation used for communication among the thesis authors.

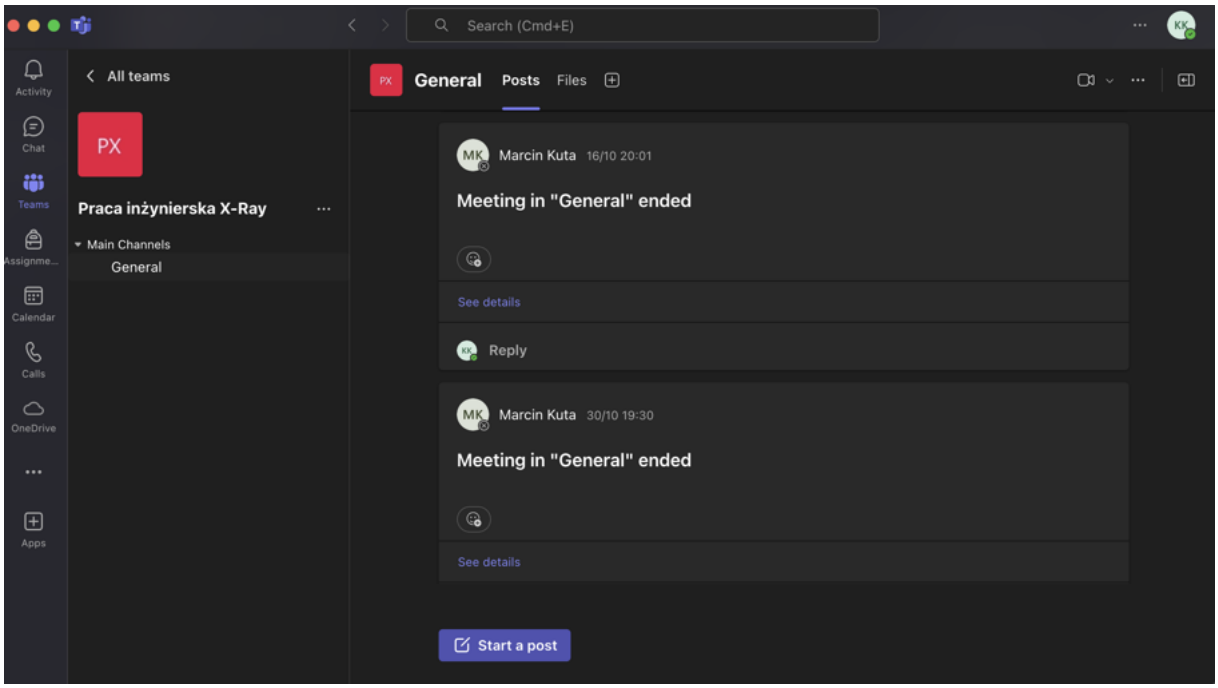

Figure 4.3: Snapshot of a Microsoft Teams conversation used for communication between the thesis team members and the Thesis Supervisor.

The entire process of software development and workflow planning was challenging. Despite periods of smooth progress and occasional delays, the team successfully created a system that meets the defined minimum requirements. The authors gained valuable knowledge and experience that will benefit them in their future academic careers or as software engineers in industry.

4.3. Division of tasks and roles

- **Michał Sośnik**
  - Implementation of the backend system.
  - Handling medical data, including the design and implementation of the database.
  - Developing the API and managing the upload of X-ray images for automatic analysis and user access.
  - Creating and supervising the remote repository on GitHub, where the application's components were developed.
  - Adding components and ensuring the security of the application.
  - Regular collaboration with other team members on aligning the backend system with the frontend and reviewing proposed changes to backend design.
- **Krzysztof Ćwiertnia**
  - Creation of system mockups.
  - Implementation of the frontend.
  - Adding chat and video call functionalities between patients and doctors.
  - Integrating frontend with backend and deep learning models.
  - Ensuring application development via code reviews.
  - Leading frontend development with regular collaboration on backend integration and proposing design adjustments based on frontend design needs.
- **Rafał Piwowar**
  - Finding datasets for training models, including X-ray wrist and arm datasets.
  - Collaborating with Kacper Kozik on the development of deep learning models.
  - Implementing federated learning with differential privacy for data privacy.
  - Conducting experiments with various levels of privacy protection (parameter ε).
  - Analyzing the impact of parameter ε on the accuracy and convergence time of the model.
  - Managing the Discord channel for communication.
- **Kacper Kozik**
  - Defining the problem, proposing the thesis topic, and providing its description.
  - Planning team responsibilities and organizing meetings with the Supervisor to report progress.
  - Defining target groups using user stories and preparing functional requirements.
  - Preparing an initial version of the database schema.
  - Proposing system mockups and defining functionalities of the application.
  - Analyzing and training Faster R-CNN and YOLO v9 models for wrist fracture detection (GRAZPEDWRI-DX dataset).
  - Conducting an analysis of the impact of model hyperparameters on detection metrics such as mAP, PR curves, precision, and recall.

4.4. Task management, collaboration and team practices

- **Organizing meetings after important milestones**

  Meetings with the Thesis Supervisor and the Coordinator of the Project Laboratory course were held after reaching key milestones. These allowed the team to review progress, address challenges, and plan the next steps.

- **Assigning tasks after meetings**

  Tasks were divided after meetings with the Supervisor or Coordinator. This ensured that the feedback could be applied immediately, giving the team time to make improvements before the next meeting.

- **Using Trello for task management**

  Trello was used to assign and track tasks. It helped everyone monitor their progress, stay organized, and keep the workflow clear.

- **Applying user stories for planning**

  User stories helped simulate real scenarios and use cases, aligning the functionalities with the project goals and providing a clear structure for task division.

- **Regular updates to technical documentation**

  Documentation was updated after finishing a set of tasks, typically a few days before meetings with the Supervisor to review new sections and receive feedback.

- **Incorporating feedback from Stakeholders**

  Feedback from meetings was used to adjust tasks and improve the application.

- **Communication software**

  The team used Trello, GitHub, and Discord for daily work and coordination. For formal discussions, Microsoft Teams was used for meetings with the Thesis Supervisor and the Coordinator of the Project Laboratory course.

4.5. Project milestones

- **February 2024**

  Initial planning for the thesis started. The project description was drafted, and the team began searching for X-ray datasets for fracture detection models. By the end of the month, the datasets for training were selected, and the project vision was created to guide further work.

- **March 2024**

  The team planned a preliminary division of tasks. Project goals were defined, and a risk analysis was conducted to identify potential challenges.

- **April 2024**

  User stories were created to outline scenarios and use cases. Target groups were identified to align the project with user needs. Work started with the analysis of the X-ray datasets and creating early dashboards, along with initial functionalities and user interactions for the application.

- **May 2024**

  Functional and non-functional requirements were defined. The team researched popular technologies for building applications and systems and selected the most suitable ones. Detailed tasks were planned for the initial development phase. By the end of the month, the first two chapters of the thesis were completed.

- **June 2024**

  Work began on static frontend components, such as basic layouts and page structures. The backend structure and database schema were also planned. Code was prepared to handle X-ray datasets, and training of models on the wrist X-ray dataset started. Research on federated learning was initiated to support the system's privacy features.

- **July 2024**

  The initial database was created, along with basic backend classes. Early implementations of messaging and video call functionalities in the application began. Helper functions for visualizing model training results from Faster R-CNN were also developed.

- **August 2024**

  Progress continued on both backend and frontend development. The team regularly reviewed code and held meetings to stay with the project's progress.

- **September 2024**

  The team worked on tuning hyperparameters for the Faster R-CNN models. Development of YOLO models started. The first prototype of the application was created, including a basic database, backend endpoints, and patient dashboard pages.

- **October 2024**

  Training of machine learning models and federated learning techniques was finalized. Doctor pages were added to the application and integrated with existing features. Messaging and video call functionality was fully implemented. The first version of the "Selected Realization Aspects" chapter was completed.

- **November 2024**

  The team gathered all results from the X-ray image analysis and models. Improvements were made to the application based on feedback from the Thesis Supervisor and Project Coordinator. New features like appointment scheduling and user login were added. Work on the final chapters of the technical documentation and initial presentation slides started.

- **December 2024**

  The MVP of the application was completed. The team held a final meeting with the Project Coordinator to showcase the application and technical documentation. The first version of presentation slides was prepared. A meeting with the Thesis Supervisor was also held to finalize the work and plan the remaining improvements.

- **January 2025**

  The thesis and final version of the application and technical documentation were completed. All preparations for the thesis submission were finalized.

Chapter 5

Project results

This chapter summarizes the project outcomes, including the developed software application, documentation, and key functionalities. The results are evaluated for usability based on performed example scenarios.

5.1. Developed application and example features

The application successfully implements all the functionalities described in chapter 3, providing a practical system for X-ray image analysis and communication between patients and doctors. Figures 5.1, 5.2, and 5.3 illustrate some of the core functionalities of the application.

- **X-ray image upload and analysis**

  Patients can upload X-ray images for fracture detection. Once uploaded, the system processes the image and marks any detected anomalies, such as fractures, directly on the image (Figure 5.1). This feature provides results, assisting both patients and doctors.

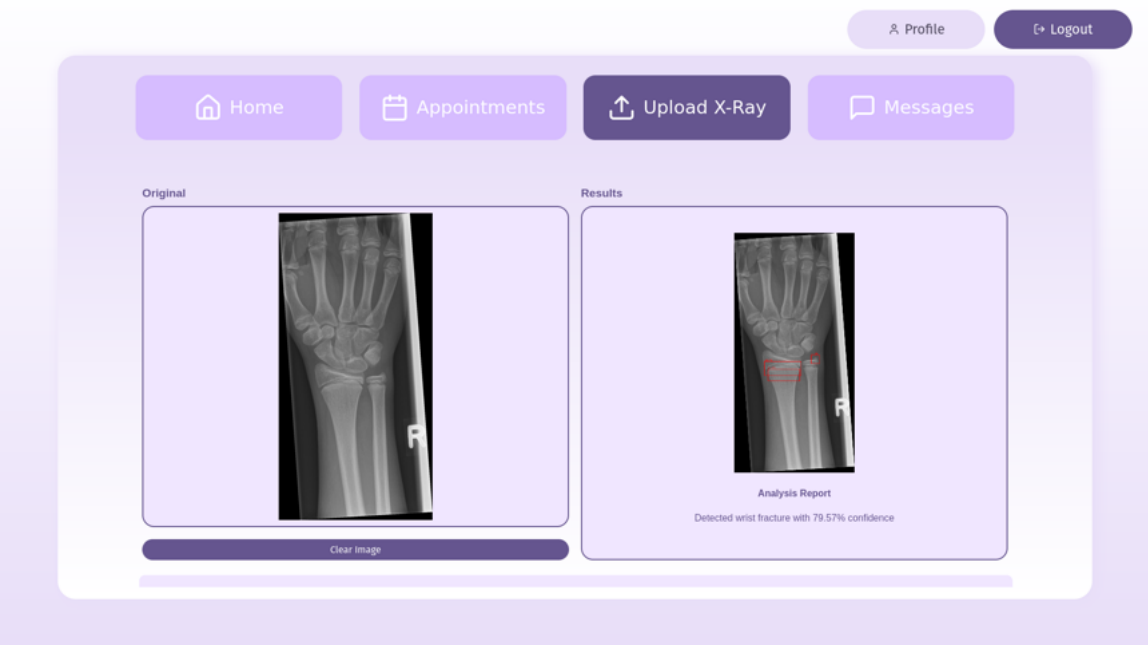

Figure 5.1: Demonstration of X-ray detection functionality in the application.

- **Doctor-Patient communication – chat system:**

  Patients and doctors can exchange messages in real time, discussing medical issues or scheduling appointments (Figure 5.2).

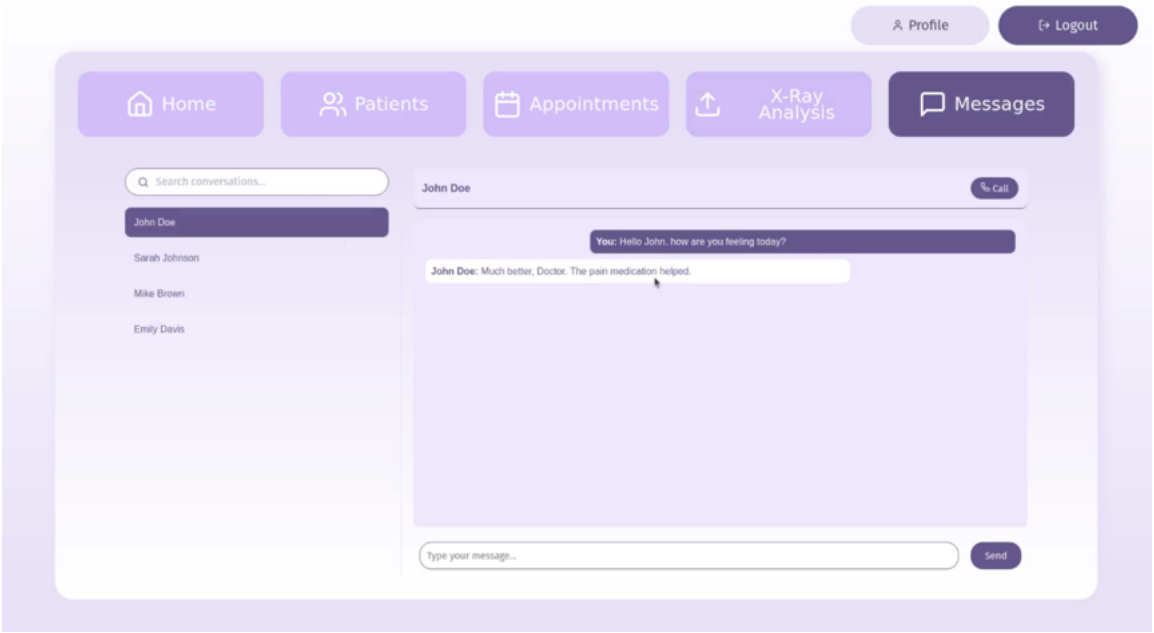

Figure 5.2: Demonstration of the messaging feature between patient and doctor in the application.

- **Video calls:**

  The video call feature allows live interactions, such as consultations or discussions, with additional options for screen sharing and video streaming (Figure 5.3).

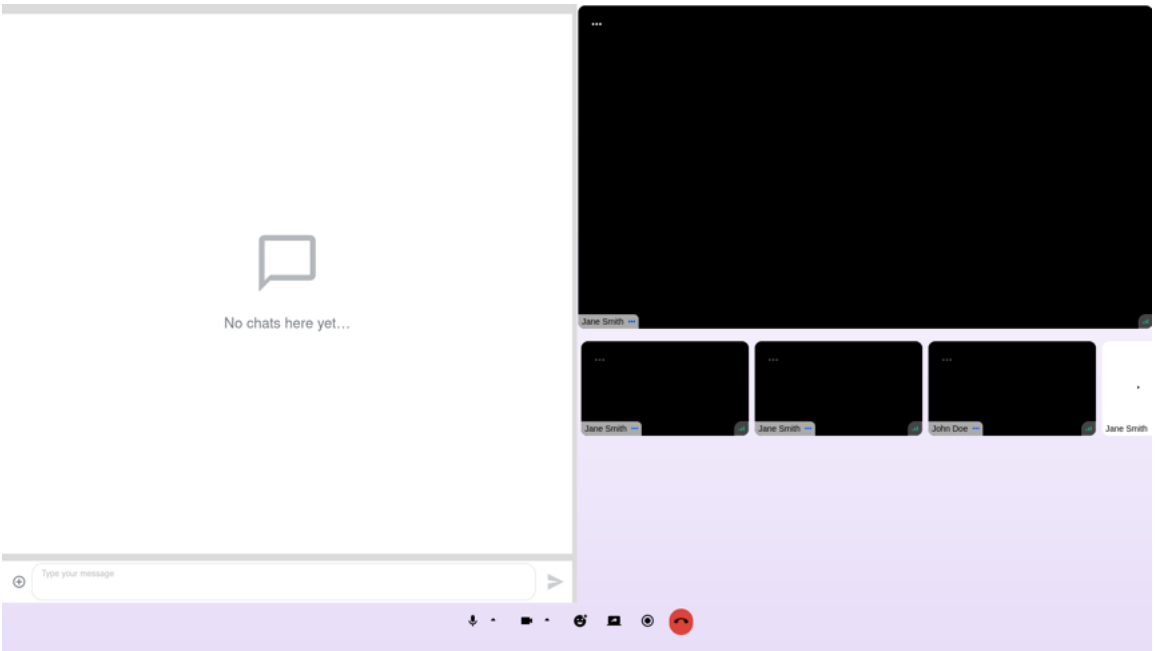

Figure 5.3: Example of video call functionality for real-time interaction between patients and doctors.

5.2. Limitations and future work

The project faced limitations, including hardware limitations that affected model training and prevented testing deeper model architectures. Training was done on a local laptop since Google Colab resources were not effective. Attempts to use them, even with paid subscription versions of Colab machines, did not significantly speed up training or support testing other models. The limited diversity of the data sets also highlighted the need for more data to improve the generalization of the model. Although Faster R-CNN and YOLO delivered promising results, challenges such as false positives and overlapping bounding boxes sometimes remain. Future work could focus on refining hyperparameters and exploring new deep learning models and architectures.

Despite these challenges, the application is ready to be deployed for image analysis, helping patients and doctors communicate, share results, and discuss medical cases. On the development side, adding more medical features, improving chat and video communication, and testing the application in different user scenarios are recommended. Expanding the system to detect anomalies in other body parts and adding model explainability features would make it even more useful in medical practice. This work provides a solid base for future improvements that can be included in the next versions of the application.

Bibliography

[1] Iowa State University. *History of Radiography*. URL: https://www.nde-ed.org/NDETechniques/Radiography/Introduction/history.xhtml.

[2] J. G. Anderson. "William Morgan and X-rays". In: *Transactions of the Faculty of Actuaries* 17 (1945), pp. 219–221.

[3] T. Wyman. "Fernando Sanford and the Discovery of X-rays". In: *"Imprint", from the Associates of the Stanford University Libraries* (2005), pp. 5–15.

[4] J. J. Thomson. "The Discharge of Electricity Through Gases". In: (1903), pp. 182–186.

[5] R. A. K. Ihor I. Mayba Roman Gaida. *Ukrainian Physicist Contributes to the Discovery of X-Rays*. 1997. URL: https://web.archive.org/web/20080528172938/http://www.meduniv.lviv.ua/oldsite/puluj.html.

[6] N. Tesla. *Incandescent electric light*. US Patent 514,170. 1894. URL: https://patents.google.com/patent/US514170.

[7] Gleamer.ai. *BoneView - Clinical AI for Bone Trauma X-rays*. URL: https://www.gleamer.ai/solutions/boneview.

[8] Alkoby. *Bone Fracture Detection App*. URL: https://github.com/Alkoby/Bone-Fracture-Detection.

[9] B. McMahan and D. Ramage. *Federated Learning: Collaborative Machine Learning without Centralized Training Data*. https://research.google.com. 2017.

[10] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao. "A survey on federated learning". In: *Knowledge-Based Systems* 216 (2021), p. 106775. DOI: 10.1016/j.knosys.2021.106775.

[11] E. Nagy, M. Janisch, F. Hržić, E. Sorantin, and S. Tschauner. "A pediatric wrist trauma X-ray dataset (GRAZPEDWRI-DX) for machine learning". In: *Scientific Data* 9 (2022), p. 222. DOI: 10.1038/s41597-022-01328-z.

[12] R. B. Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[13] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104 (2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.

[14] S. Ren, K. He, R. Girshick, and J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *arXiv: Computer Vision and Pattern Recognition* (2016). DOI: 10.48550/arXiv.1506.01497.

[15] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. "Microsoft COCO: Common Objects in Context". In: *arXiv: Computer Vision and Pattern Recognition* (2014). DOI: 10.48550/arXiv.1405.0312.

[16] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *arXiv: Computer Vision and Pattern Recognition* (2015). DOI: 10.48550/arXiv.1512.03385.

[17] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. “Feature Pyramid Networks for Object Detection”. In: *arXiv: Computer Vision and Pattern Recognition* (2016). DOI: 10.48550/arXiv.1612.03144.

[18] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese. “Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression”. In: *arXiv: Computer Vision and Pattern Recognition* (2019). DOI: 10.48550/arXiv.1902.09630.

[19] C.-T. Chien, R.-Y. Ju, K.-Y. Chou, and J.-S. Chiang. “YOLOv9 for Fracture Detection in Pediatric Wrist Trauma X-ray Images”. In: *Electronics Letters* 60 (2024), e13248. DOI: 10.1049/ell2.13248.

[20] Q. Yang, Y. Liu, T. Chen, and Y. Tong. *Federated Machine Learning: Concept and Applications*. 2019. DOI: 10.1145/3298981.

[21] J. Fu, Y. Hong, X. Ling, L. Wang, X. Ran, Z. Sun, W. H. Wang, Z. Chen, and Y. Cao. “Differentially Private Federated Learning: A Systematic Review”. In: *CoRR* 1.1 (May 2024), p. 36. DOI: 10.1007/978-3-031-72751-1_1.

[22] K. A. Bonawitz et al. “Towards Federated Learning at Scale: System Design”. In: *Proceedings of the Second Conference on Machine Learning and Systems, SysML 2019, Stanford, CA, USA, March 31 - April 2, 2019*. Ed. by A. Talwalkar, V. Smith, and M. Zaharia. mlsys.org, 2019. URL: https://proceedings.mlsys.org/paper%5C_files/paper/2019/hash/7b770da633baf74895be22a8807f1a8f-Abstract.html.

[23] User "shushu". *do1 Dataset*. https://universe.roboflow.com/shushu-pugnd/do1. Open Source Dataset. 2024.

[24] User "bone fracture". *bone fracture Dataset*. https://universe.roboflow.com/bone-fracture-du9uv/bone-fracture-yzkpo. Open Source Dataset. 2022.

[25] C. Wang, I. Yeh, and H. M. Liao. “YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information”. In: *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part XXXI*. Ed. by A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol. Vol. 15089. Lecture Notes in Computer Science. Springer, pp. 1–21. DOI: 10.48550/ARXIV.2402.13616.

[26] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin. “Albumentations: Fast and Flexible Image Augmentations”. In: *Inf.* 11.2 (2020), p. 125. DOI: 10.3390/INFO11020125.

[27] M. Joshi, A. Pal, and M. Sankarasubbu. “Federated Learning for Healthcare Domain - Pipeline, Applications and Challenges”. In: *ACM Trans. Comput. Heal.* 3.4 (2022), 40:1–40:36. DOI: 10.1145/3533708.

[28] M. S. Hossain and G. Muhammad. “Emotion recognition using secure edge and cloud computing”. In: *Inf. Sci.* 504 (2019), pp. 589–601. DOI: 10.1016/J.INS.2019.07.040.

[29] J. Shen, Y. Zhao, S. Huang, and Y. Ren. “Secure and Flexible Privacy-Preserving Federated Learning Based on Multi-Key Fully Homomorphic Encryption”. In: *Electronics* 13.22 (2024). DOI: 10.3390/electronics13224478.

[30] M. Sośnik. *OpenAPI Specification for the X-ray App API*. GitHub Repository. URL: https://github.com/msosnik/X-ray_app/blob/main/backend/data/openapi.yaml.

[31] M. Sośnik and K. Ćwiertnia. *X-ray Image Interpretation System*. GitHub Repository. URL: https://github.com/msosnik/X-ray_app/tree/main.

[32] K. Kozik and R. Piwowar. *Bone Fracture Detection*. GitHub Repository. URL: https://github.com/Kacper0199/Bone-fracture-detection-XRay.